REPORT DOCUMENTATION PAGE

Form Approved OMB NO. 0704-0188

1. REPORT DATE (DD-MM-YYYY)	2. REPORT TYPE	3. DATES COVERED (From - To)
03-08-2015	Final Report	19-Apr-2011 - 18-Apr-2015

4. TITLE AND SUBTITLE
Final Report: Mobile Assisted Security in Wireless Sensor Networks

5a. CONTRACT NUMBER
W911NF-11-1-0170

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
206022

6. AUTHORS
Ali Saman Tosun, Ali Tekeoglu, Andrew Wichmann

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES AND ADDRESSES
University of Texas at San Antonio
One UTSA Circle

San Antonio, TX 78249 -1644

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS (ES)
U.S. Army Research Office
P.O. Box 12211
Research Triangle Park, NC 27709-2211

10. SPONSOR/MONITOR'S ACRONYM(S)
ARO

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
59043-CS-REP.12

12. DISTRIBUTION AVAILIBILITY STATEMENT
Approved for Public Release; Distribution Unlimited

13. SUPPLEMENTARY NOTES
The views, opinions and/or findings contained in this report are those of the author(s) and should not contrued as an official Department of the Army position, policy or decision, unless so designated by other documentation.

14. ABSTRACT
The aim of the project is to investigate ways for integrating mobile robots to improve and reduce to complexity of providing security in wireless sensor networks. We worked on mobile assisted key management to distribute keys using mobile robots and controlled path traversal where a robot has to prove the path traversed to a base station. In addition, we worked on finding location of mobile robots to use this information in security protocols, robot coordination in case multiple robots are used and they have to be connected all the time and traversal algorithms for the mobile robots.

15. SUBJECT TERMS
mobile robots, security, sensor networks

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	15. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Ali Tosun
UU	UU	UU	UU		19b. TELEPHONE NUMBER 210-458-7663

Report Title

Final Report: Mobile Assisted Security in Wireless Sensor Networks

ABSTRACT

The aim of the project is to investigate ways for integrating mobile robots to improve and reduce to complexity of providing security in wireless sensor networks. We worked on mobile assisted key management to distribute keys using mobile robots and controlled path traversal where a robot has to prove the path traversed to a base station. In addition, we worked on finding location of mobile robots to use this information in security protocols, robot coordination in case multiple robots are used and they have to be connected all the time and traversal algorithms for the mobile robots.

We spent the last year of this project on securing multimedia devices that can be connected to wireless sensor networks and publishing the work done in earlier years. We focused on HDMI based video streaming devices that have become popular over the last two years and IP cameras. Our work in this area was experimental and we used the testbed to investigate security of HDMI based multimedia devices and IP cameras. Two PhD students supported by the project graduated recently.

Enter List of papers submitted or published that acknowledge ARO support from the start of the project to the date of this printing. List the papers, including journal references, in the following categories:

(a) Papers published in peer-reviewed journals (N/A for none)

Received Paper

TOTAL:

Number of Papers published in peer-reviewed journals:

(b) Papers published in non-peer-reviewed journals (N/A for none)

Received Paper

TOTAL:

Number of Papers published in non peer-reviewed journals:

(c) Presentations

Number of Presentations: 0.00

Non Peer-Reviewed Conference Proceeding publications (other than abstracts):

Received Paper

TOTAL:

Number of Non Peer-Reviewed Conference Proceeding publications (other than abstracts):

Peer-Reviewed Conference Proceeding publications (other than abstracts):

Received Paper

TOTAL:

Number of Peer-Reviewed Conference Proceeding publications (other than abstracts):

(d) Manuscripts

Received Paper

TOTAL:

Number of Manuscripts:

Books

Received Book

TOTAL:

Received Book Chapter

TOTAL:

Patents Submitted

Patents Awarded

Awards

Graduate Students

NAME	PERCENT_SUPPORTED	Discipline
Ali Tekeoglu	0.50	
Andrew Wichmann	0.50	
FTE Equivalent:	**1.00**	
Total Number:	**2**	

Names of Post Doctorates

NAME	PERCENT_SUPPORTED
FTE Equivalent:	
Total Number:	

Names of Faculty Supported

NAME	PERCENT_SUPPORTED	National Academy Member
Ali Saman Tosun	0.30	
FTE Equivalent:	**0.30**	
Total Number:	**1**	

Names of Under Graduate students supported

NAME	PERCENT_SUPPORTED
FTE Equivalent:	
Total Number:	

Student Metrics

This section only applies to graduating undergraduates supported by this agreement in this reporting period

The number of undergraduates funded by this agreement who graduated during this period: 0.00

The number of undergraduates funded by this agreement who graduated during this period with a degree in science, mathematics, engineering, or technology fields:..... 0.00

The number of undergraduates funded by your agreement who graduated during this period and will continue to pursue a graduate or Ph.D. degree in science, mathematics, engineering, or technology fields:..... 0.00

Number of graduating undergraduates who achieved a 3.5 GPA to 4.0 (4.0 max scale):..... 0.00

Number of graduating undergraduates funded by a DoD funded Center of Excellence grant for Education, Research and Engineering:..... 0.00

The number of undergraduates funded by your agreement who graduated during this period and intend to work for the Department of Defense 0.00

The number of undergraduates funded by your agreement who graduated during this period and will receive scholarships or fellowships for further studies in science, mathematics, engineering or technology fields: 0.00

Names of Personnel receiving masters degrees

NAME
Total Number:

Names of personnel receiving PHDs

NAME
Andrew Wichmann
Ali Tekeoglu
Total Number: 2

Names of other research staff

NAME	PERCENT_SUPPORTED
FTE Equivalent:	
Total Number:	

Sub Contractors (DD882)

Inventions (DD882)

Scientific Progress

Technology Transfer

Mobile Assisted Security in Wireless Sensor Networks

We have built a comprehensive testbed for mobile assisted sensor network research. Two mobile robots, sensors and servers that can run large scale simulations are now available for students working on the project. PhD students Ali Tekeoglu and Andrew Wichmann are partially supported by the project and they both graduated recently. Over the last year we had publications at International workshop on robotic sensor networks, IEEE 11th international conference on Mobile Ad Hoc and Sensor Systems, INFOCOM Multimedia Cloud Communication Workshop.

1 Equipments Acquired For Sensor Networks Lab

We have purchased 2 Pioneer-AT Mobile Robots for Experiments. The PIONEER 3-AT is a highly versatile four wheel drive robotic platform. Powerful, yet easy to use reliable, yet flexible, P3-AT is a popular team performer for outdoor or rough-terrain projects. P3-AT offers an embedded computer option, opening the way for onboard vision processing, Ethernet-based communications, laser, DGPS, and other autonomous functions. 8 forward and 8 rear sonar sense obstacles from 15 cm to 7 m. P3-AT's powerful motors and four knobby wheels can reach speeds of .8 meters per second and carry a payload of up to 12 kg. We have purchased 2 Memsic Classroom kits. Classroom kits are ideal for getting students up and running quickly and economically. To support 10 lab stations, a collection of 30 wireless modules, 20 sensor and data acquisition boards, 10 gateway and programming boards are included. The Classroom Kit is available in 2.4GHz. We have also purchased 3D robotics UAV to be used as an aerial node to communicate with the mobile robots on the ground.

2 Personnel Supported By The Project

Ali Tekeoglu is a PhD student working on secure multimedia delivery He defended his dissertation in summer 2015. Andrew Wichmann is working on task allocation and path planning and He defended his dissertation in spring 2015.

3 Technical Manuscripts From The Project

A couple of technical manuscripts that resulted from the project are attached to the progress report.

Coordinating Robots for Connectivity in Wireless Sensor Networks

Baris Tas and Ali Şaman Tosun
Department of Computer Science
University of Texas at San Antonio
San Antonio, TX 78249
{btas,tosun}@cs.utsa.edu

***Abstract*—Mobile robots improve scalability and performance of wireless sensor networks. Protocols for many services including data collection, localization, topology control and security in wireless sensor networks include mobile robots. Using a single robot limits the scalability and performance and multiple robots are used as a result. When multiple robots are deployed, it is desirable to have the robots connected to provide improved services. However, coordinating the robots optimally to achieve connectivity among them is not trivial. We use computational geometry techniques to achieve connectivity. We use the concept of Fréchet distance between curves to synchronize the robots for connectivity. We extend the idea of Fréchet distance to multiple curves where each curve is the path of a robot. We analyze the proposed idea theoretically and show that the theory can not be applied directly due to limitations on robot speed and speed changes. Therefore, we propose a practical approach where maximum robot speed is bounded and speed changes is limited. Simulations show that the connectivity among the robots is maintained when the robots follow the movement pattern based on the Fréchet distance between the paths of consecutive robots.**

I. Introduction

A wireless sensor network (WSN) consists of potentially hundreds of sensor nodes and is deployed in an ad hoc manner for collecting data from a region of interest over a period of time. Even though the technology is new, WSNs received an enthusiastic reception in the science community as WSNs enable precise and fine-grain monitoring of a large region in real-time. Some examples of successful large-scale deployments of WSNs to date are in the context of ecology monitoring (monitoring of micro-climate forming in redwood forests), habitat monitoring (monitoring of nesting behavior of seabirds), and military surveillance (detection and classification of an intruder as a civilian, soldier, car, or SUV).

To improve the scalability and performance of WSNs, there has been a flurry of work on employing a mobile node for data collection. The data mules [1] work exploit random movement of mobile node to opportunistically collect data from a sparse WSN. Here, the nodes buffer all their data locally, and upload the data only when the mobile node arrives within direct communication distance. Zebranet [2] system uses tracking collars carried by animals for wildlife tracking. Data is forwarded in a peer-to-peer manner and redundant copies are stored in other nodes. Shared wireless info-station model [3] uses radio tagged whales as part of a biological information acquisition system. Mobility of the mobile node is not controlled in these approaches. Mobile element scheduling (MES) work [4] considers controlled mobility of the mobile node in order to reduce latency and serve the varying data-rates in the WSNs effectively. The MES work shows that the problem of planning a path for the mobile node to visit the nodes before their buffers overflow is NP-complete. Heuristic based simple solutions are proposed to address this problem [4]–[6]. Data salmon [7] constructs a spanning tree and moves the mobile base station on this tree to optimize the cost of retrieval. To reduce the size of the path the mobile node travels, rendezvous points are used as regional collection points and the mobile node collects the data from the rendezvous points [8]. Readers are referred to [9] for using a mobile element in data collection.

Multiple robots are employed in WSNs to enhance the monitoring of an environment. Using multiple robots brings its own challenges. One challenge is to preserve the connectivity among them. Having connected robots improves the services the network offers in terms of more efficient and new protocols. Also, the connectivity among the robots leads to more secure protocols. There has been some work in the literature for providing connectivity. The connectivity between the base station (BS) and a robot exploring the environment is maintained using intermediate robots in [10], [11]. The coordinated motion planning problem where robots need to cooperate is studied [12]–[14]. A distributed algorithm that preserves the connectivity of a team of robots with limited communication among the robots is proposed in [15]. Periodic connectivity where the robots are connected at fixed intervals is introduced in [16]. Periodic connectivity is desired for scenarios where the robots explores the environment individually at some points and then regains the connectivity to share their information.

We focus on a scenario where multiple robots as a team monitor an area of interest. Our aim is to have a connectivity among the robots whose paths are pre-determined. Pre-determined paths are frequently used for monitoring an area since the use of pre-determined paths has advantages over non-determined paths. When a robot traverses a region multiple times, it can optimize its path according to the regions where a sensor exists or not. It does not have to travel to empty areas. When the mapping of the environment is known, the paths are set so that the robots avoid obstacles such as rocks and impossible regions such as lakes where robots might get stuck. Also, it is easy to estimate energy use and transmission power when the paths are known. Finally, managing multiple robots is easier when pre-determined paths are used. Also, pre-determined paths serve as a first step to solving unrestricted case where no constraints are put for mobile movement.

Fréchet distance (D_F) is used to coordinate the robots in a connected way optimally in terms of transmission range. The solution of D_F between two curves gives locations of point pairs - one point is on the first curve, and the other point is on the second curve - where the distance between the points in a pair is less than the similarity value between the curves. The robots move along the curves and the movement pattern of the robots is based on the solution pairs. D_F is a similarity measure between two curves. It is introduced by Fréchet [17]. It has been used in many applications including matching of time series in databases [18], map-matching vehicle tracking data [19], [20], moving object analysis [21], [22], speech recognition [23], and song identification [24].

The rest of the paper is organized as follows. We describe the system model in Section II. The foundations of our proposal which include the Fréchet distance is studied in Section III. Theoretical aspects of the system are shown in Section IV. We provide simulation results in Section V. Other system issues are discussed in Section VI, and we conclude with Section VII.

II. System Model

The WSN in this work consists of t sensors and n robots (r_i is a robot where i is the id of a robot and $1 \leq i \leq n$). The sensors are statically deployed in a bounded region of $A \times A$. The area possibly contains obstacles. The robots move on polygonal curves as a team, monitor the area, and retrieve information from the sensors. The paths of the robots are set by the base station (BS). After the robots collect information about the environment, they return back to the BS, and transmit their data to the BS. After the batteries of the robots are recharged, they start the next round of monitoring. The BS can set new paths to the robots. The robots are connected all the time. When this connectivity is guaranteed, efficient and more robust protocols can be designed improving the security of the services the network offers. For example, events such as fire can be recognized faster. The robot sensing the event can communicate with the other robots instantly. Moreover, attacks against the network can be minimized using multiple connected robots since more information about the environment is collected at a time.

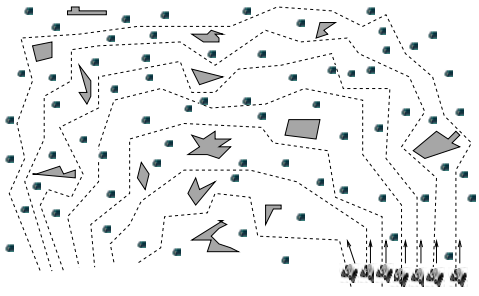


Fig. 1. System Model

Figure 1 demonstrates a possible scenario. Sensors are deployed in the area. The robots follow their pre-determined paths monitoring the environment. The polygons represent the obstacles, so the robots can not pass through them. The robots are synchronized to be connected using Fréchet distance solutions. It is also possible to have another team of connected robots monitoring and enhancing the services of the network.

III. Foundations

As a starting point for achieving the connected multiple robots, we first consider a movement pattern which guarantees the connectivity between two robots whose paths are pre-determined. Our aim is to provide the connectivity using the shortest transmission ranges.

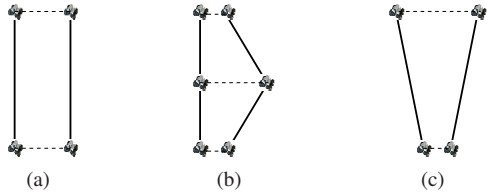


Fig. 2. How to coordinate two robots optimally.

Figure 2 gives an idea of how to coordinate two robots with optimal transmission ranges. For all the sub-figures, the robots have to move in a way so that their y-coordinates should be the same at all times to achieve the minimum distance in between. If the y-coordinates differ, they have to have longer transmission ranges to communicate. This simple observation reveals that there has to be a coordination between the robots possibly requiring changes on their speed to provide the connectivity.

A. Fréchet Distance

We use the Fréchet distance (D_F) to coordinate multiple robots. The D_F is a similarity measure between two curves. The D_F between two curves can be defined informally using an analogy to a man walking a dog on a leash. The man moves in one curve, and the dog moves in another curve. Their speed may vary, but they can not move backwards. The length of the shortest possible leash required until both complete their curves is the D_F between the curves. The D_F considers the location and ordering of the points along the curves. In our case, one robot will take the role of the man, and the other robot will take the role of the dog.

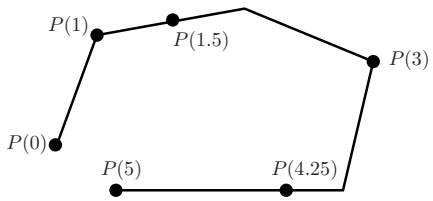


Fig. 3. Parametrization of $P : [0, 5]$

The first step to compute the D_F between two curves is to approximate a curve by a polygonal curve which is a curve entirely made up of line segments. Therefore, we mean a polygonal curve whenever we mention a curve. A polygonal curve, $P : [0, N]$, is a continuous and piecewise linear curve made up of N connected segments. N is the length of the curve. Using a parameter $a \in R$, $P : [0, N]$ can be parameterized so that $P(a)$ refers to a point on the curve. Figure 3 shows the parameterization of the curve, $P : [0, 5]$.

Time concept also needs to be parameterized to model the man-dog example. Assume that the man is following a curve $P : [0, N]$, and the dog is following a curve $Q : [0, M]$. Then,

the position of the man and the dog can be expressed as a function of t by $P(\alpha(t))$, and $Q(\beta(t))$, where $\alpha(0) = 0$, $\alpha(1) = N$, $\beta(0) = 0$, $\beta(1) = M$, and the functions α and β are continuous and non-decreasing functions.

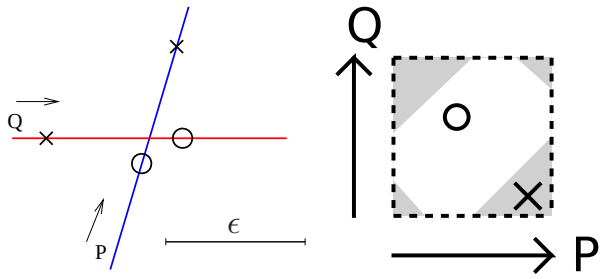
Although 2-dimensional space is considered for simplicity, the following definitions for D_F between two curves work for arbitrary dimensions. Finally, D_F is defined formally as follows [25].

Definition 1. Let V denote an arbitrary Euclidean vector space. Let $f : [a, a'] \to V$ and $g : [b, b'] \to V$ be curves where $a, a', b, b' \in R$ and $a < a', b < b'$. Then, $\mathrm{D}_F(f, g)$ denotes their Fréchet distance, defined as

$$\mathrm{D}_F(f,g) := \min_{\substack{\alpha[0,1]\to[a,a'] \\ \beta[0,1]\to[b,b']}} \max_{t\in[0,1]} ||f(\alpha(t)) - g(\beta(t))||$$

where α, β range over continuous and increasing functions with $\alpha(0) = a$, $\alpha(1) = a'$, $\beta(0) = b$, $\beta(1) = b'$.

To compute the D_F between two polygonal curves, the following decision problem is considered. Given polygonal curves P and Q, and some $\epsilon \geq 0$, decide whether $\mathrm{D}_F \leq \epsilon$. Let $P : [0, p]$ and $Q : [0, q]$ be polygonal curves where p and q are the number of edges of P and Q respectively. To solve the problem, let's first consider the case where $p = q = 1$; and define the *free space*, $F_\epsilon = (s, t) \in [0, 1]^2 \mid d(P(s), Q(t)) \leq \epsilon$, which describes all pairs of points, one on P and one on Q, whose distance is at most ϵ. Figure 4 shows line segments P, Q, a distance $\epsilon > 0$, and F_ϵ being the white area within the unit square [25]. The F_ϵ corresponding to the line segments is the intersection of the unit square with an ellipse (Its proof can be found in [25]). For example, the '$\circ$' in the white area of the free space diagram corresponds to the points represented by '$\circ$'s on the curves whose pairwise distance is less than ϵ. On the contrary, the '$\times$' in the gray area corresponds to the points represented by '$\times$'s on the curves whose pairwise distance is greater than ϵ.



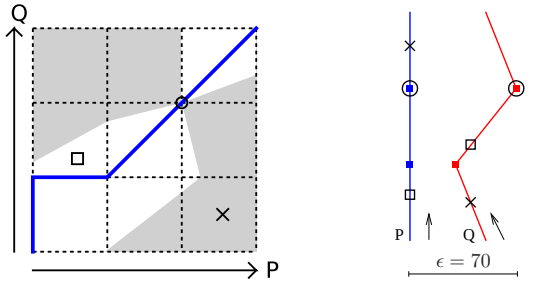
(a) One segment curves P, Q (b) Free space diagram

Fig. 4.

The F_ϵ equation is extended to arbitrary curves $P : [0, p]$ and $Q : [0, q]$ as follows:

$$F_\epsilon = (s, t) \in [0, p] \times [0, q] \mid d(P(s), Q(t)) \leq \epsilon \qquad (1)$$

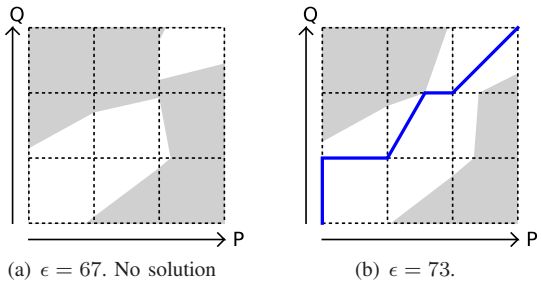
Figure 5 demonstrates an example of curves $P : [0, 3]$, $Q : [0, 3]$ along with their corresponding free space diagram for a given $\epsilon = 70$. A point on the free space diagram corresponds to a solution pair on the curves. For example, the '□' on the free space diagram corresponds to the '□'s on the curves. Similarly, the '$\circ$' corresponds to the '$\circ$'s; and the '$\times$' corresponds to the '$\times$'s on the curves. The distance between the '$\times$'s on the curves is 110, the distance between the '$\circ$'s on the curves is 70, and the distance between the '□'s is 52. The distances between the points (□, $\circ$) which correspond to the points in the white area in the free space diagram are less than the ϵ value, whereas the distance between '$\times$'s is greater than ϵ since the corresponding '$\times$' in the free space diagram is in the gray area. If there exists a monotone curve in both directions from $(0, 0)$ to (p, q) within the F_ϵ of curves P and Q, we have $\mathrm{D}_F \leq \epsilon$. In other words, the man following one curve can walk his dog following the other curve with a leash of length ϵ referring back to the man-dog example. The monocity condition comes from the fact that the man and the dog can not move backwards. If the monocity condition is eliminated, then the problem is called *weak* Fréchet Distance.



(a) Free space diagram ($\epsilon = 70$) (b) Curves: P, Q

Fig. 5. Fréchet Distance between curves.

In order to solve the decision problem, F_ϵ values are calculated starting from $\epsilon = 0$ to the smallest ϵ value which make F_ϵ contain a monotone curve from $(0, 0)$ to (p, q). At that point, D_F is found to be the final value of ϵ. However, a more clever algorithm is achieved using the technique of *parametric search* on some critical ϵ values. The runtime of this algorithm is $O(pq \log(pq))$, and its details can be found in [25]. The corresponding free space diagrams of the curves P and Q from Figure 5 for ϵ values 67, 70 and 73 are shown in Figures 5, and 6. As seen from the figures, F_ϵ gets larger as ϵ values increase, and D_F of the curves is 70 since the smallest ϵ value which make the free space diagram contain a monotone curve from $(0, 0)$ to $(3, 3)$ is 70. We call the monotone curve a solution curve for the curves P and Q ($S^\epsilon_{P,Q}$).

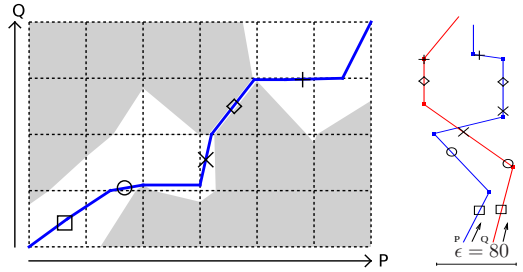


(a) $\epsilon = 67$. No solution (b) $\epsilon = 73$.

Fig. 6. Free space diagrams with different epsilon values.

B. Adapting Fréchet Distance to Connectivity Problem

We start with the basic case where the coordination between two robots is considered. Later, we extend the idea to the case where there exist arbitrary number of robots. A solution curve (S^ϵ) to the Fréchet distance between two curves is used in coordinating two robots to have a connectivity between the robots. The curves are assumed to be the paths of the robots. Let one curve be P, and the other curve be Q. Also, assume that R_P is the robot following curve P, and R_Q is the robot following curve Q. A point on $S^\epsilon_{P,Q}$ in the free space diagram of two curves corresponds to a pair, which we call *a solution pair*, consisting of two locations on the curves. One location is on curve P and the other location is on curve Q. We know that the distance between the locations in a solution pair is less than D_F, and this is also true for all solution pairs. Therefore, if the transmission ranges of the robots are greater than D_F, the robots are guaranteed to be connected at the solution locations. As a result, a movement pattern for the robots is driven based on the solution locations.



(a) Free space diagram ($\epsilon = 80$) (b) P, Q

Fig. 7. Robot movement based on Fréchet Distance.

Since the paths of the robots are pre-determined, D_F and the corresponding free space diagram can be computed. Figure 7 shows two paths $P[0:N]$, $Q[0:M]$ and their corresponding free space diagram where $N = 6$ is the size of curve P, $M = 4$ is the size of curve Q. Consider the point '□' on curve P. Let $\text{Param}_{(P,\square)}$ be the parametrized value of the point '□' (see Figure 3 for curve parametrization). The intersection of the line segment (($\text{Param}_{(P,\square)}$, 0); ($\text{Param}_{(P,\square)}$, M)) and the solution curve in the free space diagram is represented as '□' in Figure 7(a). The x-coordinate of this point is $\text{Param}_{(P,\square)}$ and the y-coordinate is $\text{Param}_{(Q,\square)}$ where $\text{Param}_{(Q,\square)}$ is the parametrized value of the point '□' on curve Q. Using this value, it is trivial to compute the xy-coordinates of the point '□' on curve Q. Similarly, '○', '×', '◇', and '+' are the other example *solution pair*s on curves P and Q. The distances between the points in each pair are less than D_F. If the robots, R_P and R_Q, are coordinated so that they reside at the pair points at the same time, they are guaranteed to be connected at these points. For example, assume that R_P with constant speed passes by the locations '□', '○', '×', '◇', and '+' on curve P at times $t_\square$, $t_\circ$, $t_\times$, $t_\diamond$, and t_+ respectively. If the speed of R_Q is arranged so that R_Q passes by the locations '□', '○', '×', '◇', and '+' on curve Q at times $t_\square$, $t_\circ$, $t_\times$, $t_\diamond$, and t_+ respectively, then it is guaranteed that the robots are connected at the specified points. In theory, any point on a curve and its pair point on the other curve can be computed using the solution curve on the free space diagram and continuous connectivity is achieved.

We extend the case where there exist two robots to the case with arbitrary number of robots. When multiple robots are coordinated, we assume a simplified version where the paths of the robots do not intersect (the case where the paths intersect can be dealt with solving Fréchet distance for partial curves). In this way, the idea used for two robots can be applied to every two consecutive robots in a chained way. Assume that there exist three robots, R_P, R_Q, and R_R following the curves $P[0:5]$, $Q[0:6]$, $R[0:4]$ respectively as shown in Figure 8(a). Consider the first two consecutive robots R_P and R_Q. The free space diagram of the curves P and Q ($FSD_{(P,Q)}$) is given in Figure 8(b). Using the solution curve in $FSD_{(P,Q)}$, solution pairs on the curves can be computed as discussed in the two robot case. For example, for the locations '□', '○', '×', '◇', and '+' on curve P; their corresponding locations on curve Q derived from the solution curve in $FSD_{(P,Q)}$ is shown on curve Q. To compute the corresponding points on curve R, we make use of $\text{Param}_{(Q,\square)}$, $\text{Param}_{(Q,\circ)}$, $\text{Param}_{(Q,\times)}$, $\text{Param}_{(Q,\diamond)}$ and $\text{Param}_{(Q,+)}$ along with $FSD_{(Q,R)}$. For instance, $\text{Param}_{(R,\square)}$ is the intersection of the line segment $((\text{Param}_{(Q,\square)}, 0); (\text{Param}_{(Q,\square)}, 4))$ and the solution curve in $FSD_{(Q,R)}$. Once $\text{Param}_{(R,\square)}$ is found, computing the xy-coordinates of the point '□' on curve R is trivial. Finally, when the speeds of the robots V_{R_P}, V_{R_Q}, and V_{R_R} are arranged so that the robots reside at points '□' on their curves at time $t_\square$, the connectivity is achieved at locations '□'s on the curves. The same also applies for the '○', '×', '◇', '+' locations.

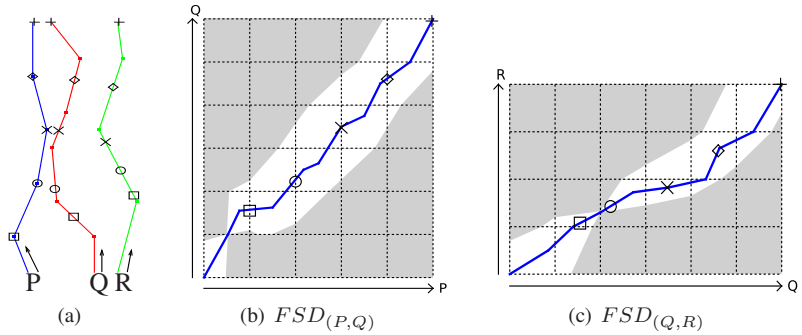
1) Coordinating Multiple Robots in an Application: Theoretically, multiple robots are guaranteed to be connected at all times when all solutions from the free space diagram is used. However, this might cause too many changes on the speed of the robots. We explain this in detail in section V. To limit the speed changes for robots, we coordinate the robots based on one robot (R_{base}). Only the vertices on the path of R_{base} are considered for the resulting solution points. For example, consider that robots R_P, R_Q and R_R is set to follow the curves P, Q, R in Figure 8(a). If R_P is picked as R_{base}, then the vertices of curve P are considered as solution points and the corresponding solution points on the other curves are computed as discussed above. This results in connectivity when the robots are at locations □, ○, ×, ◇, + on their own curves. Any robot could have been chosen as R_{base}. If the application requires more locations where the robots are connected, additional robots are picked as R_{base}, and the solutions from each setting are combined.

IV. Theoretical Foundations

In this section, we discuss the theoretical aspects of our idea. We will show that the speed of the mobile robots needs to be updated during traversal to follow the solution curve. Consider the following example.

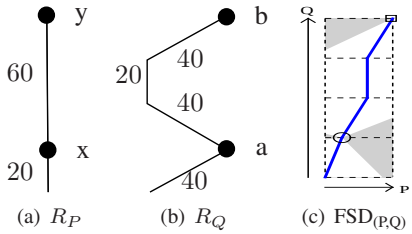
In Figure 9, when robot R_P is at x, robot R_Q must be at a and when robot R_P is at y, robot R_Q must be at b. In the first part, when R_P travels 20 units, R_Q travels 40 units and in the second part when R_P travels 60 units, R_Q travels 100 units. So, even if R_P has constant speed, R_Q needs to update its speed during traversal.

Theorem 1. *Given curves C_1 and C_2, a solution curve S^ϵ in Free Space diagram $FSD_{(C_1,C_2)}$, mobile robots can follow*



(a) (b) $FSD_{(P,Q)}$ (c) $FSD_{(Q,R)}$

Fig. 8. Multiple robot movement.

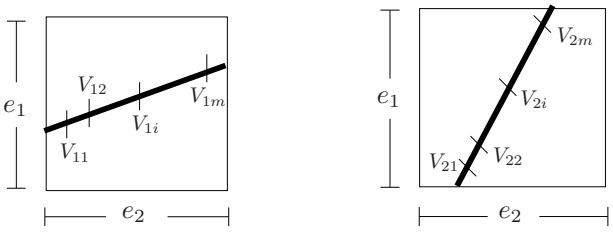


(a) R_P (b) R_Q (c) $\text{FSD}_{(P,Q)}$

Fig. 9. The speed of the robots needs to be updated

the solution curve S^ϵ by setting the speed on curve C_2 relative to the speed on curve C_1.

Proof: Solution curve S^ϵ intersecting current cell of free space diagram can be represented as a special case of one of the following.



(a) (b)

Fig. 10. Example

Figure 10(a) follows a complete edge of C_2 and Figure 10(b) follows a complete edge of C_1. Let corresponding edges on curves C_1 and C_2 be e_1 and e_2 with lengths $|e_1|$ and $|e_2|$ respectively. Let the left and right endpoints of solution in current cell be $L_1 = (x_1, y_1)$ and $L_2 = (x_2, y_2)$ respectively.

- *Case 1:* Current cell of free space diagram follows a complete edge of C_2 as in left side of figure. Let fraction of e_1 in current cell be α.
  Line segment $[L_1, L_2]$ may have multiple segments with different speed requirements on C_1. Let's assume there are m segments and let $V_{1i}, 1 \leq i \leq m$ be the speed on segment i. Since time spend on both C_1 and C_2 must be equal for each segment, we have

$$\frac{V_{1i}}{V_{2i}} = \frac{\alpha|e_1|}{|e_2|}$$

So, speed on C_2 for a line segment with speed V_{1i} on C_1 is

$$V_{2i} = V_{1i}\frac{|e_2|}{\alpha|e_1|}$$

- *Case 2:* Current cell of free space diagram follows a complete edge of C_1 as in left side of figure. Let fraction of e_2 in current cell be β.
  Line segment $[L_1, L_2]$ may have multiple segments with different speed requirements on C_2. Let's assume there are m segments and let $V_{1i}, 1 \leq i \leq m$ be the speed on segment i. Since time spend on both C_1 and C_2 must be equal for each segment, we have

$$\frac{V_{1i}}{V_{2i}} = \frac{|e_1|}{\beta|e_2|}$$

So, speed on C_2 for a line segment with speed V_{1i} on C_1 is

$$V_{2i} = V_{1i}\frac{\beta|e_2|}{|e_1|}$$

□

We use the following terminology for multiple curves. Given k curves $C_1, \ldots, C_k$ and a pairwise solution curves $S^\epsilon_{i,i+1}$ is a solution curve for curve C_i and C_{i+1} and free space diagram $FSD^\epsilon_{i,i+1}$ is the free space diagram for curves C_i and C_{i+1}.

Theorem 2. *Given k curves $C_1, \ldots, C_k$ and pairwise solution curves $S^\epsilon_{i,i+1}, 1 \leq i \leq k-1$ in free space diagram $FSD^\epsilon_{i,i+1}, 1 \leq i \leq k-1$, we can follow all solution curves $S^\epsilon_{i,i+1}, 1 \leq i \leq k-1$ simultaneously.*

Proof: By repeated application of Theorem 1. Follow solution curve $S^\epsilon_{i,i+1}$ by setting the speed on curve C_{i+1} relative to the speed on curve C_i. □

Depending on the pattern of the robot paths, some heuristics can be applied. However, these heuristics do not yield the optimal solution. For instance, plane sweep solution would fail in the following example. Consider Figure 11.

Plane sweep in this example yields a distance of 4. However, Fréchet distance in this case is $2\sqrt{2}$.

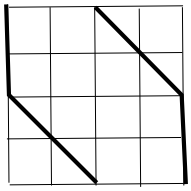


Fig. 11. Example

V. Simulations

We conducted simulations to support the proposed method both in terms of theory and practice to verify if multiple robots can have connectivity in a wireless network. We implemented the simulation using ns2 network simulator [26]. 802.15.4 MAC layer is used for low-rate wireless communications. In addition to the simulation results presented in this paper, supplementary videos created from the simulations can be found on the project web page [27].

n random polygonal curves covering an area of 1000×1000 are created for n robots. The curves are generated so that they do not intersect. Robot R_i follows curve c_i. The x-coordinate of any vertices of c_i is less than the x-coordinate of any vertices of c_{i+1}. Simulations with up to 20 robots are conducted. To test the connectivity, we kept track of the number of packets sent and received between consecutive robots. The robots send dummy packets to their neighbors with a frequency of f which is set to 1.0s. To mitigate collisions, we added a back-off mechanism. For R_i, a duration of $\frac{f}{n} * i$ is added to the time R_i sends its packet. Let $\mathrm{D}_F(c_i, c_{i+1})$ be the Fréchet distance between curves c_i and c_{i+1}. The transmission range of a robot, R_i, is set to $\max(\mathrm{D}_F(c_{i-1}, c_i), \mathrm{D}_F(c_i, c_{i+1})) + \lambda$ where $\lambda = 10$m.

The type of a simulation where the solution curve between the robots R_1 and R_2 is used as S^{ϵ}_{base} is called sim_{first}. Also, the type of a simulation where the solution curve between the robots $R_{\frac{n}{2}}$ and $R_{\frac{n}{2}+1}$ is used as S^{ϵ}_{base} is called sim_{mid}. In the second approach to find the solution points, we use each solution curve as S^{ϵ}_{base} and combine all resulting solution points. We call this type of simulations sim_{all}. We study both the theoretical and the practical versions.

A. Continuous Connectivity

This section simulates the theoretical solution. One of the solution curves is picked as the base solution curve $S^{\epsilon}_{base} = S^{\epsilon}_{(1,2)}$. $S^{\epsilon}_{(1,2)}$ is the solution curve between the first and the second robot. Solution points for the first robot is selected by intersecting $S^{\epsilon}_{(1,2)}$ with the free space diagram grid (FSDG). FSDG is defined as the horizontal segments:

$$hl_0[(0,0);(N,0)], hl_1[(0,1);(N,1)], \ldots, hl_M[(0,M);(N,M)]$$

and the vertical segments:

$$vl_0[(0,0);(0,M)], vl_1[(1,0);(1,M)], \ldots, vl_N[(N,0);(N,M)]$$

on the free space curve where N is the curve size of c_1 and M is the curve size of c_2. A segment is represented as the xy-coordinates on the free space diagram. Once we find the solution points for $S^{\epsilon}_{(1,2)}$, we find the corresponding solution points on the rest of the solution curves. The solution points found on $S^{\epsilon}_{(1,2)}$ is converted to the locations on the curve, c_1. We set a constant speed, Speed_1, for R_1 which follows c_1. Using Speed_1, the associated time values, t_p, are found for all the solution points. In this way, all curves have to adapt their speed to reside at the corresponding solution points at the same time based on R_1's speed. For example, while R_1 is moving at constant speed, R_2 has to adapt its speed at the solution points based on $S^{\epsilon}_{(1,2)}$. R_2 moves at constant speed between the solution points. For c_2, there exists additional solution points from the intersection of the solution curve between c_2 and c_3; and FSDG of free space diagram of c_2 and c_3. The timing values for these additional points are found based on R_2's speed and they are used as speed change locations for R_3. For the rest of the robots, the same idea is used causing additional speed change locations for each robot.

In Figure 12, we see a snapshot taken from a simulation using the theoretical approach where there are five robots, and the outer robot has the constant speed 3m/s. A circle represents the transmission range of a robot.

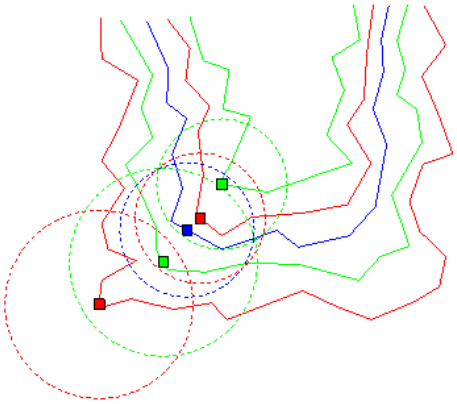


Fig. 12. Snapshot for the simulation with five robots.

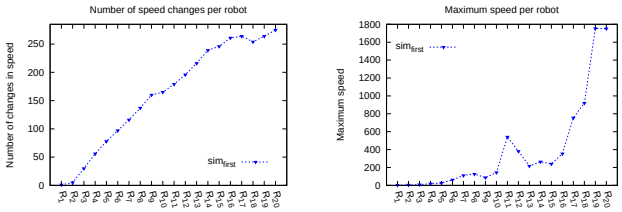
When the theoretical approach is used, the connectivity is guaranteed all the time. Table I shows the connectivity results for the above example with 5 robots when $f = 1.0$s. For each robot, R_i, the number of packets R_i broadcasts, the number of packets received from the previous robot R_{i-1}, and the number of packets received from the next robot R_{i+1} is shown as a row in the table.

TABLE I. Robot Connectivity Results (Theory)

R_i	Sent	Received from R_{i-1}	Received from R_{i+1}
R_1	348	-	348
R_2	348	348	348
R_3	348	348	348
R_4	348	348	347
R_5	347	348	-

Figure 13(a) demonstrates the number of speed change per robot when R_1 moves at constant speed 3m/s for 20 robots. The last robot has to change its speed 275 times.

We also analyzed the maximum speed a robot can take for the same setting. Figure 13(b) shows the maximum speed each robot takes. To guarantee the continuous connectivity a robot should travel with the speed of 1753m/s. Since this is not applicable with the current technology, we propose a practical approach by setting a maximum speed a robot can take.



(a) (b)

Fig. 13. Based on theory. R_1 has constant speed 3m/s.

B. Connectivity at Selected Points

Since the maximum speed a robot takes is not applicable in the theoretical approach, we set a limit on the speed a robot can take for practical reasons. In this approach, connectivity is guaranteed at selected points. The solution points where connectivity is guaranteed are found as follows. In the first approach, one of the solution curves is picked as the base solution curve S^{ϵ}_{base}. On this solution curve, the points on every Δ distance are chosen as the solution points. Δ values we use are $0.4, 0.2, 0.1$ and 0.05. The corresponding solutions points on the rest of the solution curves are found using these solution points. Detailed discussion about finding the corresponding solution points of the other solution curves based on one solution curve can be found in subsection III-B.

The speeds of the robots are arranged as follows. Let's assume that all robots reside at their current solution points at time t_k. For example, R_i resides at $L_{(R_i,t_k)}$ at time t_k. Each robot knows the location ($L_{(R_i,t_{k+1})}$) to be at time t_{k+1} to have connectivity at t_{k+1}. Simulation takes an argument, $maxSpeed$, which specifies the maximum speed a robot can travel with. The robot which needs to travel farthest to its next solution point will adjust its speed to $maxSpeed$. The maximum distance to next solution point is found as:

$$maxDistance_k = \max_{1 \le i \le n} ||L_{(R_i,t_{k+1})} - L_{(R_i,t_k)}||$$

Let's assume that a^{th} robot needs to travel farthest to its next solution point. Then, the speed of R_a is set to $maxSpeed$. Once, the speed of R_a is found, t_{k+1} is computed; and the speeds of the rest of the robots (R_i where $1 \le i \le n$ and $i \neq a$) are adjusted using t_k, t_{k+1} and $||L_{(R_i,t_{k+1})} - L_{(R_i,t_k)}||$. So the speed of the robot R_i is set to:

$$S_i = \frac{||L_{(R_i,t_{k+1})} - L_{(R_i,t_k)}||}{t_{k+1} - t_k}$$

Figure 14 demonstrates a few snapshots from a simulation run of type sim_{first} with 20 robots, $f = 1.0$s, and $maxSpeed = 10$. A circle represents the transmission range of a robot. The simulation takes 452s, and the snapshots are taken at seconds 150 and 300 respectively.

Table II presents the connectivity results for ten robots when $f = 1.0$s, $maxSpeed = 10$m/s and the type of the simulation is sim_{first}. Almost all of the packets a robot sends will be received by the neighboring robots, and a robot receives almost all of the packets the neighboring robots broadcast. Some robots can not receive all the packets a neighboring robot sends (see the communication between R_5 and R_6 in the table). Although the robots are guaranteed to be connected at the solution points, we do not guarantee that whether a robot broadcasts at the solution points since a robot broadcasts dummy packets with a frequency of 1.0s in our simulation. However, even in this case, only a few packets are missing which shows that the network is connected almost all the time. Figure 15 shows a case where the connectivity is lost. R_P is guaranteed to be connected with R_Q at consecutive locations (P_1, Q_1) and (P_2, Q_2). However, there is no locations that guarantee connectivity in between $|P_1, P_2|$ and $|Q_1, Q_2|$. As a result, connectivity is lost.

TABLE II. ROBOT CONNECTIVITY RESULTS FOR 10 ROBOTS

R_i	Sent	Received from R_{i-1}	Received from R_{i+1}
R_1	376	-	376
R_2	376	375	376
R_3	376	376	376
R_4	376	376	376
R_5	376	376	370
R_6	376	372	376
R_7	376	376	375
R_8	375	376	375
R_9	375	375	375
R_{10}	375	375	-

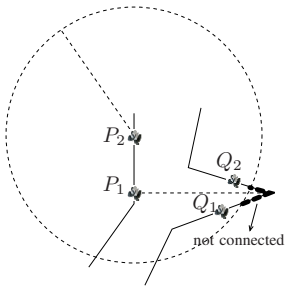


Fig. 15. Not connected

TABLE III. ROBOT CONNECTIVITY RESULTS FOR 5 ROBOTS

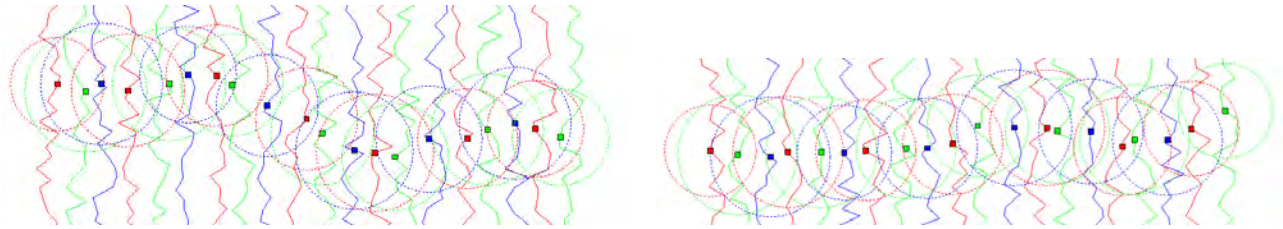
R_i	Sent	Received from R_{i-1}	Received from R_{i+1}
R_1	345	-	345
R_2	345	345	345
R_3	345	345	344
R_4	344	345	339
R_5	344	338	-

Table III shows the results for five robots where $\Delta = 0.4$. Moreover, Table IV shows the results for five robots where $\Delta = 0.4$ on curves which are more skewed and consist of more segments. Table V shows the results for five robots following the same type of curves where $\Delta = 0.05$. All of them are results from type sim_{first} simulation.

TABLE IV. ROBOT CONNECTIVITY RESULTS FOR 5 ROBOTS

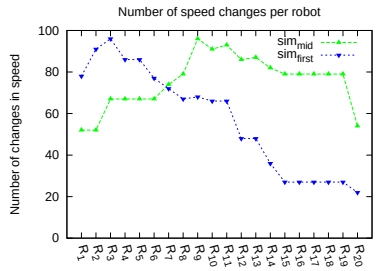
R_i	Sent	Received from R_{i-1}	Received from R_{i+1}
R_1	492	-	491
R_2	491	492	487
R_3	491	485	490
R_4	491	479	477
R_5	491	479	-

Figure 16 shows the number of changes in speed per robot when the solution points are selected based on the curves' segment end points. x-axis denotes the robots and y-axis denotes the number of changes in speed. Figure 16(a) compares the simulation types sim_{first} and sim_{mid}. In this approach, R_{base} has the most number of speed changes since

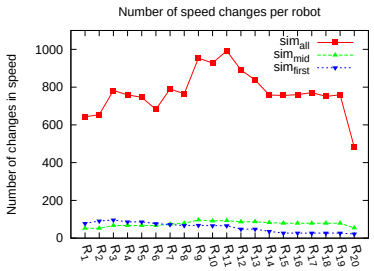


(a) $t = 150$ (b) $t = 300$

Fig. 14. Snapshots from our simulation. 20 robots



(a) First and middle robot.



(b) Comparison of all settings.

Fig. 16. Comparison of the settings with respect to number of speed changes.

TABLE V. ROBOT CONNECTIVITY RESULTS FOR 5 ROBOTS

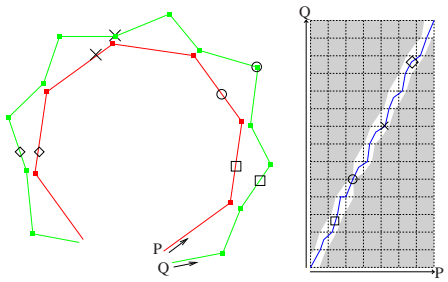
R_i	Sent	Received from R_{i-1}	Received from R_{i+1}
R_1	505	-	504
R_2	504	505	504
R_3	504	504	501
R_4	504	498	491
R_5	504	492	-

the vertical and horizontal lines on the rest of the solution curves cause solution point losses. Figure 16(b) compares all of the simulation types sim_{first}, sim_{mid} and sim_{all}. Since there exist more solution points in sim_{all} type, the speed of a robot needs to be adjusted many more times compared to the simulation types, sim_{first} and sim_{mid}. Finally, the total numbers of changes in speed for sim_{first}, sim_{mid} and sim_{all} are 1142, 1509, and 15460 respectively.

VI. DISCUSSION

It is easy to develop heuristics based on the movement pattern of the robots for an application. If robots are moving in one direction, plane sweep heuristic can be used. However, this approach can not be generalized. For example, Figure 17 shows a case where plane-sweep heuristic can not be used. Also, even if the robots move in one direction, plane sweep heuristic does not yield the optimum solution in terms of transmission range. Figure 18 compares the transmission ranges for Fréchet distance and plane sweep when the robots are moving in one direction. Fréchet distance solution serves as a general solution to the problem of connectivity among robots for any kind of path pattern given.

We focused on paths which do not cross in this paper. However, proposed idea can be applied to crossing paths with a modification. The paths can be split at the crossing points and proposed idea can be used for partial curves with different order of curves on each side of crossing point. Figure 19 demonstrates the idea. At point, C, the curves Q and R intersect. Curves are split by the dashed line. The



(a) (b)

Fig. 17. Fréchet as a general solution

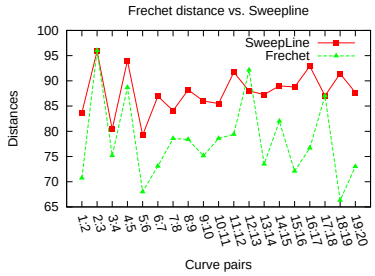


Fig. 18. Frechet Distance vs. Sweepline

solution points for the curves below the dashed line and the solution points for the curves above the dashed line are found separately using the proposed idea. Below the dashed line we have solution points corresponding to P and Q and Q and R. Although same order of curves can be used above the crossing point, distances will end of being large. Instead we can use solution points corresponding to P and R and R and Q above the dashed line.

In proposed approach, we set the transmission range of a robot based on the Fréchet distance and use the same range during the traversal of the path. Since the paths are

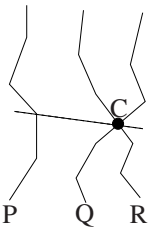


Fig. 19. Crossing paths

pre-determined, the transmission range of the robots can be adjusted dynamically to save energy and reduce interference. Robot can compute the range it needs to have to reach the other robots and use this range for communication.

VII. Conclusion

Mobile robots are introduced to overcome the limitations of wireless sensor networks. When multiple robots are used, maintaining connectivity between mobile robots is challenging. We propose a method to guarantee connectivity among multiple robots which move in a coordinated manner. Having continuously connected robots is important since it improves the efficiency and scalability of network protocols. In addition, larger area can be monitored at a time and the network becomes more responsive to attacks and events when continuous connectivity is achieved. We extend the idea of Fréchet distance between two curves to compute the Fréchet distance between several curves and provide a practical implementation of the approach. We provide a theoretical analysis of Fréchet distance to form the foundations of our approach. We simulate the proposed method and verify the connectivity among the robots using our method. Proposed approach is generic and can be applied to a large number of robot movement patterns.

References

[1] R. C. Shah, S. Roy, S. Jain, and W. Brunette, "Data mules: modeling a three-tier architecture for sparse sensor networks," in *Proceedings of the First IEEE International Workshop on Sensor Network Protocols and Applications*, 2003, pp. 30–41.

[2] P. Juang, H. Oki, and Y. Wang, "Energy-efficient computing for wildlife tracking: Design tradeoffs and early experiences with zebranet," in 10^{th} *International Conference on Architectural Support for Programming languages and Operating Systems.*, October 2002.

[3] T. Small and Z. Haas, "The shared wireless infostation model - a new ad hoc networking paradigm (or where there is a whale, there is a way)," in *ACM MobiHoc*, 2003, pp. 233–244.

[4] A. Somasundara, A. Ramamoorthy, and M. Srivastava, "Mobile element scheduling for efficient data collection in wireless sensor networks with dynamic deadlines," in *Proceedings of the 25th IEEE International Real-Time Systems Symposium*, 2004, pp. 296–305.

[5] Y. Gu, D. Bozdag, E. Ekici, F. Ozguner, and C. Lee, "Partitioning based mobile element scheduling in wireless sensor networks," in *IEEE SECON*, 2005, pp. 386–395.

[6] W. Zhao and M. Ammar, "Message ferrying: Proactive routing in highly-partitioned wireless ad hoc networks," in *Proceedings of the The Ninth IEEE Workshop on Future Trends of Distributed Computing Systems*, 2003, pp. 308– 314.

[7] M. Demirbas, O. Soysal, and A. S. Tosun, "Data salmon: A greedy mobile basestation protocol for efficient data collection in wireless sensor networks," in *IEEE International Conference on Distributed Computing in Sensor Systems*, 2007.

[8] G. Xing, T. Wang, Z. Xie, and W. Jia, "Rendezvous planning in mobility-assistedwireless sensor networks," in *28th IEEE International Real-Time Systems Symposium*, 2007, pp. 311–320.

[9] M. Di Francesco, S. K. Das, and G. Anastasi, "Data collection in wireless sensor networks with mobile elements: A survey," *ACM Trans. Sen. Netw.*, vol. 8, no. 1, pp. 7:1–7:31, Aug. 2011.

[10] *2008 IEEE International Conference on Robotics and Automation, ICRA 2008, May 19-23, 2008, Pasadena, California, USA*. IEEE, 2008.

[11] D. Tardioli, A. Mosteo, L. Riazuelo, J. Villarroel, and L. Montano, "Enforcing network connectivity in robot team missions," *Int. J. Rob. Res.*, vol. 29, no. 4, pp. 460–480, Apr. 2010. [Online]. Available: http://dx.doi.org/10.1177/0278364909358274

[12] M. Defoort and K. Veluvolu, "A motion planning framework with connectivity management for multiple cooperative robots," *Journal of Intelligent & Robotic Systems*, pp. 1–15, 2013. [Online]. Available: http://dx.doi.org/10.1007/s10846-013-9872-0

[13] O. Dengiz, A. Konak, and A. E. Smith, "Connectivity management in mobile ad hoc networks using particle swarm optimization," *Ad Hoc Netw.*, vol. 9, no. 7, pp. 1312–1326, Sep. 2011. [Online]. Available: http://dx.doi.org/10.1016/j.adhoc.2011.01.010

[14] M. Defoort, A. Kokosy, T. Floquet, W. Perruquetti, and J. Palos, "Motion planning for cooperative unicycle-type mobile robots with limited sensing ranges: A distributed receding horizon approach," *Robot. Auton. Syst.*, vol. 57, no. 11, pp. 1094–1106, Nov. 2009. [Online]. Available: http://dx.doi.org/10.1016/j.robot.2009.07.004

[15] N. Michael, M. M. Zavlanos, V. Kumar, and G. J. Pappas, "Maintaining connectivity in mobile robot networks," in *ISER*, 2008, pp. 117–126.

[16] G. Hollinger and S. Singh, "Multi-robot coordination with periodic connectivity," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, 2010, pp. 4457–4462.

[17] M. Fréchet, "Sur quelques points du calcul fonctionnel," *Rendiconti del Circolo Matematico di Palermo (1884 - 1940)*, vol. 22, pp. 1–72, 1906.

[18] M.-S. Kim, S.-W. Kim, and M. Shin, "Optimization of subsequence matching under time warping in time-series databases," in *SAC*, 2005, pp. 581–586.

[19] S. Brakatsoulas, D. Pfoser, R. Salas, and C. Wenk, "On map-matching vehicle tracking data," in *Proceedings of the 31st international conference on Very large data bases*, ser. VLDB '05. VLDB Endowment, 2005, pp. 853–864. [Online]. Available: http://dl.acm.org/citation.cfm?id=1083592.1083691

[20] C. Wenk, R. Salas, and D. Pfoser, "Addressing the need for map-matching speed: Localizing global curve-matching algorithms," in *Scientific and Statistical Database Management, 2006. 18th International Conference on*, 2006, pp. 379–388.

[21] K. Buchin, M. Buchin, and J. Gudmundsson, "Detecting single file movement," in *Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems*, ser. GIS '08. New York, NY, USA: ACM, 2008, pp. 33:1–33:10. [Online]. Available: http://doi.acm.org/10.1145/1463434.1463476

[22] K. Buchin, M. Buchin, J. Gudmundsson, M. Löffler, and J. Luo, "Detecting commuting patterns by clustering subtrajectories," in *Proceedings of the 19th International Symposium on Algorithms and Computation*, ser. ISAAC '08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 644–655. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-92182-0_57

[23] S. Kwong, Q. He, K.-F. Man, C. W. Chau, and K.-S. Tang, "Parallel genetic-based hybrid pattern matching algorithm for isolated word recognition," *IJPRAI*, vol. 12, no. 4, pp. 573–594, 1998.

[24] J. Serrà, E. Gómez, P. Herrera, and X. Serra, "Chroma binary similarity and local alignment applied to cover song identification," *IEEE Transactions on Audio, Speech & Language Processing*, vol. 16, no. 6, pp. 1138–1151, 2008.

[25] H. Alt and M. Godau, "Computing the fréchet distance between two polygonal curves," *Int. J. Comput. Geometry Appl.*, vol. 5, pp. 75–91, 1995.

[26] S. McCanne and S. Floyd, "ns network simulator," http://www.isi.edu/nsnam/ns/.

[27] B. Tas and A. S. Tosun, "Project web page for simulation results," www.cs.utsa.edu/~tosun/robotconnectivity/robotcon.html, accessed: 06 April 2014.

Resilient Data Collection in Mobile-assisted Wireless Sensor Networks

Baris Tas and Ali Şaman Tosun
Department of Computer Science
University of Texas at San Antonio
San Antonio, TX 78249
email{btas,tosun}@cs.utsa.edu

ABSTRACT

Mobility facilitates efficient data collection protocols improving the performance, scalability and life-time of wireless sensor networks. We propose a simple, yet effective and scalable method for resilient data collection in mobile-assisted wireless sensor networks. The mobile element covers an area using periodic long-range broadcast messages. Upon receiving a broadcast message, a sensor sends its data to the mobile element using trajectory routing in a multi-hop manner. The mobile element includes sensor acknowledgements in the broadcast using a Bloom filter. If a packet is not received by the mobile element due to an erroneous node along the trajectory, a different trajectory is used to avoid malicious nodes. Simulation results demonstrate that low number of broadcasts is enough to collect data from a large-scale network with over 99% success rate if the system parameters are set properly.

1. INTRODUCTION

To improve the scalability and performance of WSNs, there has been a flurry of work on employing a mobile node for data collection. The data mules [12] work exploit random movement of mobile node to opportunistically collect data from a sparse WSN. Here, the nodes buffer all their data locally, and upload the data only when the mobile node arrives within direct communication distance. Zebranet [4] system uses tracking collars carried by animals for wildlife tracking. Data is forwarded in a peer-to-peer manner and redundant copies are stored in other nodes. Shared wireless info-station model [13] uses radio tagged whales as part of a biological information acquisition system. Mobility of the mobile node is not controlled in these approaches. Mobile element scheduling (MES) work [14] considers controlled mobility of the mobile node in order to reduce latency and serve the varying data-rates in the WSNs effectively.

WSNs are vulnerable to various attacks due to their nature. In selective forwarding, a sensor on the path from the source to the destination drops forwarding packets [5]. Proposed solutions for detection of the attack include watchdog mechanisms where a node keeps track of its neighbors' behavior [8]. However, this solution depletes sensors' resources quickly. Another scheme which uses acknowledgement from intermediate nodes is proposed in [16]. False forwarding, where a node does not follow the forwarding mechanism precisely, is a type of misrouting attack. A malicious node falsify the routing packets to disrupt the routing tables [6]. In the wormhole attack, an adversary tunnels messages received in one part of the network over a low-latency link and replays them in a different part. An adversary could convince nodes who would normally be multiple hops from a base station that they are only one or two hops away via the wormhole [5]. To defend against wormhole attacks, a leash is added to a packet to restrict the packet's maximum allowed transmission distance [3].

The main components of our data collection protocol include trajectory routing, a cone-based topology control mechanism, and Bloom filter. Trajectory-based routing (TBR) described in [10, 9] is a generalization of source based routing, and cartesian routing. In TBR, a packet is forwarded along a curve set by the source. A cone-based distributed topology control mechanism proposed in [7] preserves the network connectivity by ensuring at least one neighbor exists in every cone of degree α around each sensor. Bloom filter is a space-efficient randomized hash-coding method for representing a set to support membership queries introduced by Burton Bloom [1]. These components fit together in harmony producing a scalable and efficient data collection mechanism.

We propose a data collection mechanism resilient to the node failures or packet dropping attacks for *large-scale* mobile-assisted wireless sensor networks. The WSN we consider consists of a mobile element (ME) and static sensors. The ME covers the area to be monitored using a pre-determined route. Since an ME is expected to have more resources than a regular sensor, its transmission range can be longer than a sensor's. On its journey, the ME broadcasts *long-range* messages. Each broadcast message triggers the sensors in the vicinity of the ME to reply back with their data in a multi-hop manner using trajectory routing. The cone-based topology control mechanism is employed to control the number of hops a packet travels towards the ME. Bloom filter is the main data structure of our system's acknowledgement mechanism. Simulation results indicate the ME is able to collect data from over 99% of the total sensors using the proposed scheme.

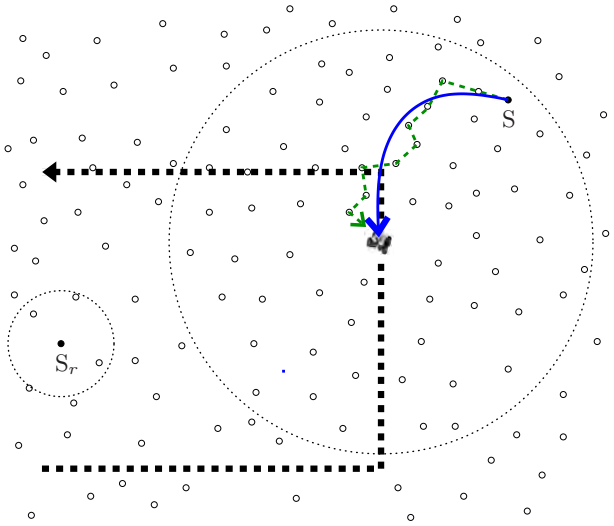


Figure 1: System Model

2. SYSTEM MODEL

The WSN in this work consists of an ME and n sensors (s_i is a sensor where i is the id of the sensor and $1 \leq i \leq n$). The sensors are statically deployed in a bounded region of $A \times A$. We assign the transmission range of a sensor according to the distances between the sensor and its close neighbors with a cone-based topology control mechanism. This approach reduces the transmission interference which is a bottleneck for the performance of an application designed for a dense WSN. Also, it extends the life-time of the sensors. r_i is the range of the sensor with the id i; whereas r_{ME} is the range of the ME. The range of the ME is the same throughout the application except for a few initial transmissions. Also, It is greater than the range of any sensors. The ME covers the area of interest with periodic broadcasts, and speed, V_{ME}. Each broadcast message is associated with a sequence number (SN_i: i^{th} sequence number). Sensors within the transmission range of a broadcast reply back to the ME with their data in a multi-hop manner. The ME follows a space-filling curve as its route. Once the ME completes its tour, it reports all the sensor readings it has collected to the base station (BS).

The communication from the sensors to the ME is based on trajectory routing. Trajectory routing requires a dense network, and the nodes know the locations of their neighbors, at least approximately. Therefore, the sensors are assumed to know their locations and the locations of their neighbors. Also, the ME knows the sensor locations approximately. These can be achieved using mobile-assisted localization techniques such as the one proposed in [11], or an expensive option for localization would be attaching a GPS device for each node. In Trajectory routing, the source node embeds a curve into the packet, and the intermediate nodes forward the packet as close as possible to the curve using greedy techniques.

The system model is shown in Figure 1. The ME collects data from the static sensors deployed in an area of interest. The circles represent the sensors. The thick dashed line represents the route of the ME. The transmission range of the ME is larger than the transmission range of a sensor. The transmission range of a sensor (S_r), and the ME is depicted in the figure. The ME transmits long-range broadcasts. The broadcast message triggers the sensors within the broadcast range to send their data. An acknowledgement mechanism bypasses unnecessary replies. For example, when the sensor, S, receives the broadcast message, it sends its data along the curve as shown in the figure if S has not received its acknowledgement yet. The intermediate nodes forward the packet originated from S to the ME along the trajectory shown as the dashed line.

3. PROPOSED SCHEME

The ME covers the area to be monitored following a space-filling curve. It transmits broadcast messages. A broadcast message has two functionalities. It triggers the sensors within the vicinity of the ME to reply back with their data, and it contains a Bloom filter carrying the acknowledgements for the sensors whose data have been successfully received after the previous broadcast. Trajectory routing is used when the sensors send their data to the ME. Collisions are mitigated using the acknowledgement mechanism, the cone-based topology control algorithm, and a back-off mechanism which adds proper delays to the reply messages from the sensors to the ME. The communications are secured using symmetric-asymmetric keys and cryptographic functions. The proposed scheme is designed for dense networks, and it is resilient to node failures and packet dropping attacks.

3.1 Sensor-ME Communication

Trajectory routing is used for the communication from the sensors to the ME. When a sensor receives a broadcast message, it either drops the packet, or replies back to the ME depending on whether the sensor had previously sent its data to the ME *successfully*, or not. If the ME has not received the sensor data yet, the sensor picks a trajectory where the starting point of the trajectory is the sensor location, and the final point of the trajectory is the ME position. Then, the sensor embeds the trajectory into the packet, and the packet is sent to the ME along the trajectory in a multi-hop manner.

Although a broad range of curves can be defined, we pick the upper or lower arc of the major axis of an ellipse as the trajectory. When a sensor is ready to send its data to the ME, it picks one side of the ellipse, embeds that trajectory into its packet, and sends the packet. If it receives an acknowledgement at the next broadcast, the sensor is done with sending its own data to the ME; and drops the succeeding broadcast messages. However, it continues to forward the packets which were originated by the other sensors if the sensor is along the curve of those packets. If the sensor does not receive the corresponding acknowledgement at the next broadcast due to node failures, collisions, or insufficient density of the network, it picks the *other* side of the ellipse as its trajectory; and sends its data along this trajectory. This approach increases the probability of the data packets to be received by the ME. The major axis of the ellipse divides the plane in half, and we guarantee that a node does not forward its packet if its only forwarding choice is a node on the other half plane. Since the half planes can not contain a sensor in common, the hops along both curves are different.

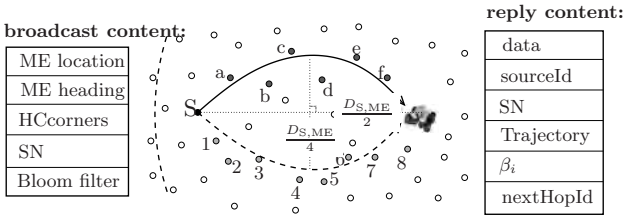


Figure 2: Switching between curves

Figure 2 shows the interaction between an ME and sensor nodes. In the figure, circles represent sensor nodes, the two arcs (solid and dashed) represent two different trajectories. The dashed arc on the left shows the long-range broadcast of the ME since the figure shows a tiny portion of the whole network. The rectangles show the broadcast and reply message contents. The source node, S, embeds the solid trajectory into its packet and sends the packet to the ME in a hop-by-hop manner. The packet follows the trajectory, and in the ideal case, the packet is transmitted to the ME through the sensor nodes a, b, c, d, e, and f. If the acknowledgement for this packet is not received at the next broadcast, the sensor, S, sends its data along the dashed trajectory. The packet is expected to be forwarded through the sensors 1, 2, 3, 4, 5, 6, 7, 8.

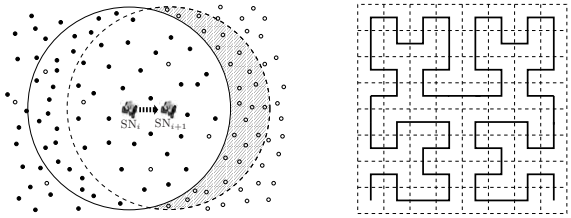
An *acknowledgement* mechanism is required to prevent a sensor from replying to every broadcast message it receives. Each broadcast message has a Bloom filter containing the acknowledgements corresponding to successful sensor data receptions triggered by the previous broadcast message. In this way, sensors become aware of their successful data transmissions. With the acknowledgement mechanism, node failures on the trajectories can also be detected. Moreover, if a sensor does not receive its acknowledgement due to a node failure on the trajectory, it uses a different trajectory increasing the resilience to the node failures. Finally, the acknowledgement mechanism increases the overall quality of the network bypassing unnecessary transmissions.

Since the ME uses long-range broadcast messages, it is possible that a sensor receives the broadcast message multiple times with different ME locations. As a result, the sensor uses a trajectory having different destination location. In this way, different intermediate nodes are used for the sensor-ME communication when the ME is at different regions increasing the resilience to the node failures. As a result, A sensor will have many chances to transmit its data to the ME thanks to the long-range broadcasts and separate forwarding paths increasing the resilience to node failures.

3.2 ME-Sensor Communication

The ME transmits long-range broadcast messages periodically. The broadcast period, τ, is an important factor for the performance of our system. A significant issue regarding the broadcast period is the limitation on the packet size for WSNs. Remember that we use an acknowledgement mechanism to prevent further replies from the sensors whose data had been received successfully by the ME previously. To achieve this, the acknowledgements are embedded to the broadcast packets. Because of the packet size limitation, there is a limit on the number of the acknowledgements that can be embedded into the broadcast packet. We use a probabilistic space-efficient data structure, Bloom filter, to increase the number of acknowledgements that can be embedded into a broadcast message.

Bloom filter is used to test whether an element is a member of a set or not. It is a one-bit vector array of size m, initially all bits set to 0. Whenever an element is inserted to the Bloom filter, the element is first hashed by k independent and uniformly distributed hash functions. Each of the k hash values is used as the bit index of the Bloom filter which is set to 1. In our protocol, when the ME receives a packet originating from a sensor, s_i, with the sequence number SN_j, the ME applies the one-way *sha1* function on $\mathrm{id}_{\mathrm{s_i}}|\mathrm{SN}_j$. The Bloom filter is reset for each broadcast message. *sha1* produces a 20 byte message digest of which we use each of the first k bytes as the index values to the Bloom filter. This is repeated for all the sensors which can transmit their data successfully to the ME. The Bloom filter is included in the next broadcast message (SN_{j+1}). When the sensor, s_i, receives the broadcast, it queries the Bloom filter using the result of $sha1$ applied on $\mathrm{id}_{\mathrm{s_i}}|\mathrm{SN}_j$. The sensor either drops the broadcast or replies back to the ME depending on the membership in the Bloom filter. If the sensor does not see its id in the Bloom filter, it replies with its data. In the meantime, the ME resets the Bloom filter for the new broadcast, and inserts the acknowledgement for the sensors from which the ME received their data. The bloom filter containing the acknowledgement corresponding to the broadcast message with SN_{j+1} is included in the new broadcast message with SN_{j+2}. The same mechanism is repeated for all broadcasts. In this way, a sensor knows whether the sensor data is received by the ME or not.



(a) Consecutive broadcast messages

(b) Hilbert curve

Figure 3: Data collection protocol.

The false positives rate (*fpr*) of a Bloom filter can be controlled through the parameters n_{bf}, m, and k; since the *fpr* is approximately $(1-e^{\frac{-kn}{m}})^k$. Because of the packet size limitation in WSNs, we pick m as 256 bits (32 bytes). Moreover, we only let 1% or less than 1% of *fpr*. Under these constraints, n_{bf} is found as 25, and k is found as 5 after trying various values for n_{bf} and k for our data collection application. As a result, our Bloom filter is capable of holding n_{bf} acknowledgements for n_{bf} sensors at each broadcast message. n_{bf}, the number of total sensors in the network (n), the speed (v_{ME}) and the range of the ME (r_{ME}) are the key factors for determining the broadcast period. Assuming most of the sensors within the range of the previous broadcast message are acknowledged, the next broadcast is transmitted when there are n_{bf} expected unacknowledged sensors within the range of the ME. In Figure 3(a), the solid circles represent the sensors which have received their ac-

knowledgements from the ME; and the void circles represent the sensors which could not receive their acknowledgements yet. The figure shows two consecutive broadcasts. Since the Bloom filter is able to carry $n_{\rm bf}$ sensor acknowledgements, the next broadcast is transmitted when the shaded area in the figure holds $n_{\rm bf}$ expected number of sensors. Let the shaded area be $\rm A_s$. Then, $\rm A_s = \frac{n_{bf}}{n} A^2$ where A^2 is the total area of the monitored field. A_s is actually the difference of the two circles with radius r_{ME}: $A_s = C_{SN_{i+1}} - C_{SN_i}$ where C_{SN_i} is the transmission area of the broadcast message with SN_i, and $C_{SN_{i+1}}$ is the transmission area of the consecutive broadcast message. From geometry, we know that the area of the difference of any two circles with the same radius is $\rm Circ_{diff} = \pi r_{ME}^2 - 2(\frac{\theta r_{ME}^2}{2} - \frac{d}{2}\sqrt{r_{ME}^2 - (\frac{d}{2})^2})$ where $\theta = 2\arccos(\frac{d}{2r_{ME}})$ is the angle of the arc between the intersection points of the two circles, and d is the distance between the two circles. If $A_s == \rm Circ_{diff}$, then the centers of the circles are the locations of two consecutive broadcast messages. Let d_c be the d value guaranteeing the equality, $A_s == \rm Circ_{diff}$. d_c becomes the expected distance between two consecutive broadcast messages. Since all the variables other than d is known, d_c is computed using a binary search. Then, the expected broadcast period, τ, is calculated as $\tau = \frac{d_c}{V_{ME}}$.

3.3 Controlling Collisions

Upon receiving a broadcast message, if the sensors in the vicinity of the ME reply back to the ME all at the same time, *collision*s occur inevitably. Therefore, a *back-off mechanism* is required to mitigate the number of collisions. After finding the delay, a sensor *predict*s the location of the ME at the time (current time + delay) the sensor will be sending its data, and uses this location as the destination point of its trajectory.

The aim of the back-off mechanism is to assign different periods of delays to the sensors. After receiving a broadcast message, a sensor, s_i, calculates a delay of period ($Delay_{s_i}$) which is less than the broadcast period, τ. $Delay_{s_i}$ depends on the orientation of the sensor with respect to the ME, and the heading of the ME. Recall that two consecutive broadcasts intersect because of the limitation on the packet size. This limitation favors the mitigation of the collisions since fewer sensors have to send their data within the period between two consecutive broadcasts.

The location of the ME at the time a sensor is sending its reply message needs to be predicted. The predicted ME location is set as the destination point of the trajectory. The broadcast message includes the ME location, and a few corners of the space filling curve being used. For example, if Hilbert Curve (HC) [2] is used as the route of the ME, the next two HC corners the ME will visit are included in the broadcast. The speed of the ME is also known by the sensor (can be included in the broadcast messages). Using this information together with the $Delay_{s_i}$, predicting the location of the ME at the time s_i sends its reply message becomes trivial.

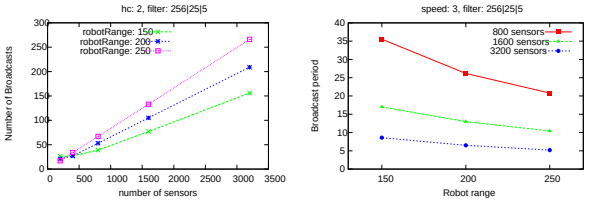
4. SIMULATIONS

We conducted extensive simulations to support the proposed scheme. We implemented the simulations using ns2 wireless network simulator. Our protocol is implemented and added as a new protocol to ns2 core code. We used a modified version of *802.11* MAC layer. The RTS/CTS mechanism is disabled to provide the communication between two elements which have different transmission ranges as our protocol requires, and to mimic *802.15.4* standards. Disabling the RTS/CTS protocol also mitigates the collisions since there will be relatively less transmissions. In addition to the simulation results presented in this paper, supplementary simulation results can be found on the project web page [15]. Videos are available for some of the individual sample runs of the simulations to give an idea about the mechanics of the system.

n sensors are deployed using uniform distribution in a region of 1000m $\times$ 1000m. The values of the system parameters are the following: number of sensors (n: 200, 400, 800, 1600, 3200), maximum sensor range (r_{max}: 10m, 20m, 30m, ..., 150m), the range of the ME (r_{ME}: 150m, 200m, 250m), cone alpha values (α: 60°, 90°,120°, 150°), curve levels (2, 3), and the ME speed (V_{ME}: 3m/s, 6m/s). Also, the constant parameters for the Bloom filter are $m = 256$, $n_{bf} = 25$, and $k = 5$. For each setting, we generated 100 different sensor configurations. Our main performance criterion is the *success rate* which is described as the percentage rate of the number of the sensors whose data is received by the ME over the number of all sensors considering that the ME tours the area to be monitored once.

Although any space-filling curve can be used as the route of the ME, we used two different space-filling curves for comparison purposes: Hilbert curve(HC), and snake-scan curve (SC) which is a non-recursive space-filling curve. A level-3 HC is shown in Figure 3(b). A HC is defined recursively. 4 level k curves are combined to have a level $k+1$ curve as follows. A square is initially divided into 4 ordered quadrants and a *first-order curve* is drawn by connecting the center of the quadrants. For the next level of HC, each of the quadrants is divided into 4 and 4 scaled-down level 1 HCs are connected by changing the level 1 HCs' orientations preserving their order. SC produces better results since it contains less number of turns compared to HC. Although the use of SC outperforms the use of HC, their success rates are almost identical. We focus on HC for the route of the ME as we present the simulation results.

The required number of broadcasts is feasible although the system achieves high success rates. The number of broadcast messages depends on r_{ME}, n, and the length of the ME route (HC_{level}). The relation among these variables are shown in Figure 4(a) where $HC_{level} = 2$ is used. As n increases, the number of broadcasts increases since the required area of the difference of two consecutive broadcast messages gets smaller and the broadcast frequency increases. Also, the number of broadcasts is directly proportional to r_{ME}. The broadcast period, τ, depends on r_{ME}, n, and V_{ME}. The relation among τ, r_{ME}, and n is demonstrated in Figure 4(a) when $V_{ME} = 3$. As r_{ME} increases, τ decreases since the area of the difference of two consecutive broadcast messages increases faster when r_{ME} is longer. When the network gets denser, the required area of the difference of two consecutive broadcast messages gets smaller since dense network has more sensors per unit of area; and τ decreases.

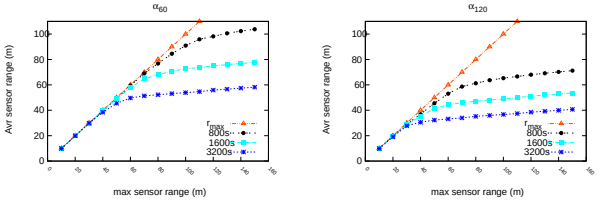


(a) (b)

Figure 4: Number of broadcasts and broadcast period

The ranges of the sensors are arranged using the cone-based topology algorithm. The technology can only allow limited number of range levels for the transmission range of the commercial sensors [7]. In our simulations, the cone-based topology mechanism is applied assuming the sensors are able to set one of eight range levels to their transmission range. r_{max} is divided into eight and the result is set to range level 1. The other range levels are computed by adding the division result each time as the eighth level being r_{max}.

Most of the simulation results we present discuss the cases where the sensor range is increasing. The graphs use the maximum sensor range (r_{max}) as the sensor values for clear representation. However, the real values for the sensor ranges are different depending on the α cone value, and the network density since we are deploying the cone-based topology algorithm. We use $\alpha = 60°$, and $120°$ in our simulations. To give an idea regarding the relation between the r_{max} values and the average of real sensor ranges for different settings, Figures 5(a), and 5(b) are used respectively for α values $60°$, and $120°$. The figures include the deployments with 800, 1600, and 3200 sensors. As seen in the figures, for low r_{max} values, the averages of the real sensor ranges are the same as the r_{max} values. As the maximum sensor range increases, the effect of cone-based topology control algorithm can be seen since a sensor now has more neighbors around it. Also, the average of the real sensor range values for dense networks is less than the average of the ones for scarce networks because of the same reason. Finally, the average of the real sensor ranges when $\alpha = 120°$ is less than the average of the real sensor ranges when $\alpha = 60°$ as expected.

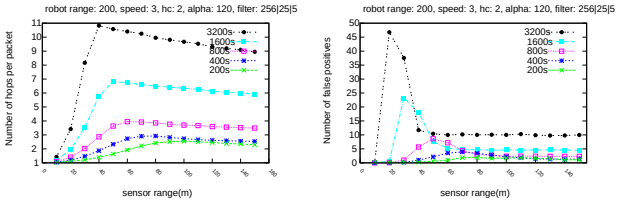


(a) α_{60} (b) α_{120}

Figure 5: Cone-based topology.

The success rate of our system is promising collecting data from over 99% of the sensors deployed. The success rate values as the maximum sensor range increases are depicted in Figure 6(a) for the configuration where $r_{ME} = 200$m, $HC_{level} = 2$, $\alpha = 120°$, and $V_{ME} = 3$m/s. In the figure, the x-axis represents the maximum sensor range. The actual sensor ranges are different than the presented since the cone-based topology is used. For example, a setting with 3200 sensors achieve the optimum success rate when $r_{max} = 40$m. The corresponding average sensor range is found to be about 30m when Figure 5(b) is analyzed. In the beginning, as the sensor range increases, the success rate also increases. The reason is when the sensor range is small, a sensor does not have enough neighbors to apply the trajectory routing. Also, dense networks achieve their optimum success rates earlier than the scarce networks because dense networks have more number of neighbors which results in better routing performance.

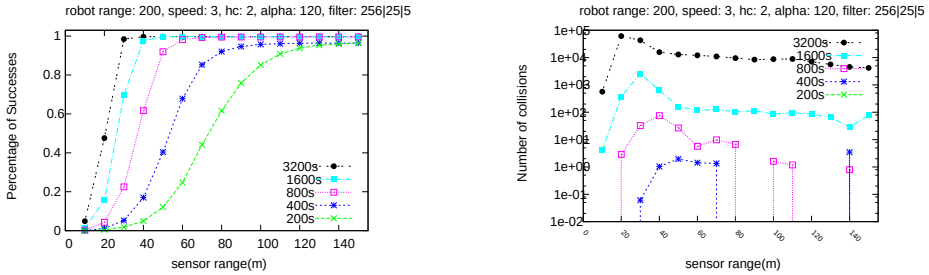
The number of total collisions as the sensor range increases for the same configuration is shown in log-scale in Figure 6(b). For all sensor range values, more collisions occur in dense networks compared to scarce networks as expected. In the beginning, the sensor range is small, so the collisions are less likely to occur. As the sensor range increases, the number of collisions also increases until sensors start to send their data to the ME successfully. When more packets are being received by the ME, the number of collisions start to decrease since a sensor is not required to re-send its data once it receives its acknowledgement. The number of collisions continue decreasing sharply until optimum success rates are achieved. Further increase in sensor range provides less collisions since the number of hops in the communications between the sensors and the ME will decrease. We also analyzed the distribution of the collisions. The maximum number of collisions per sensor is as low as 1 for individual simulation optimum settings with 1600 or less sensors. The maximum number of collisions per sensor is about 40 for the simulations resulting optimum success rate with 3200 sensors. Also, more packets collide on the sensors closer to the ME route. When the ME changes its direction, more collisions occur. Furthermore, when the range of the ME increases, the number of collisions also increases since more sensors are triggered to send their replies.



(a) (b)

Figure 7: Average number of hops per packet and total no of false positives.

The average number of hops a packet visits (Hop_{avr}) is shown in Figure 7(a). As the sensor range increases, the packets are more likely to be received by the ME and Hop_{avr} increases. Hop_{avr} reaches its maximum value when the success rate is maximum. After this point, the increase in sensor range results in less Hop_{avr} values since the packet can reach the ME using less many hops. The sensor ranges are shorter when denser networks are considered as a result of the cone-based topology algorithm. Therefore, Hop_{avr} values for dense networks are higher than Hop_{avr} values for sparse networks. The total number of false positives caused by the Bloom filter is depicted in Figure 7(b). The number



(a) Success ratio (b) Collisions

Figure 6: Success ratio and collisions.

of false positives reaches its local minimum value when the success rate is maximum. In terms of speed, the success rate decreases as the ME travels faster since the performance of the back-off mechanism decreases causing more collisions.

5. CONCLUSION

In this paper, we propose an efficient, scalable, and resilient data collection protocol for large-scale mobile assisted WSNs. The system is built on components including trajectory routing, cone-based topology mechanism, and Bloom filter which are seamlessly integrated. Performance of the proposed scheme depends on many tunable parameters including sensor range, density and topology of the network. Using simulations, we show that the system parameters can be tuned for a high rate of successful data collection. It is possible to cover a large area using limited number of broadcasts for networks with sufficiently high density ideal for trajectory routing. Collisions play an important role on the success rate, and they are minimized with a back-off mechanism. As sensor range is increased, success rate initially increases and starts to decrease after reaching its maximum value. This is due to the higher number of collisions that occur with increased range. Finally, future work includes identifying the misbehaving or malicious nodes.

6. REFERENCES

[1] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13:422–426, 1970.

[2] D. Hilbert. *Uber die stetige Abbildung einer Linie auf Flachenstuck*, volume 38, pages 459–460. Math. Ann, 1891.

[3] Y.-C. Hu, A. Perrig, and D. Johnson. Packet leashes: a defense against wormhole attacks in wireless networks. In *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, volume 3, pages 1976–1986 vol.3, 2003.

[4] P. Juang, H. Oki, and Y. Wang. Energy-efficient computing for wildlife tracking: Design tradeoffs and early experiences with zebranet. In 10^{th} *International Conference on Architectural Support for Programming languages and Operating Systems.*, October 2002.

[5] C. Karlof and D. Wagner. Secure Routing in Wireless Sensor Networks: Attacks and Countermeasures. *Elsevier's AdHoc Networks Journal, Special Issue on Sensor Network Applications and Protocols*, 1(2–3):293–315, September 2003.

[6] I. Khalil. Mimi: Mitigating packet misrouting in locally-monitored multi-hop wireless ad hoc networks. In *Global Telecommunications Conference, 2008. IEEE GLOBECOM 2008. IEEE*, pages 1–5, 2008.

[7] L. Li, J. Halpern, P. Bahl, Y.-M. Wang, and R. Wattenhofer. A cone-based distributed topology-control algorithm for wireless multi-hop networks. *Networking, IEEE/ACM Transactions on*, 13(1):147 – 159, feb. 2005.

[8] S. Marti, T. J. Giuli, K. Lai, and M. Baker. Mitigating routing misbehavior in mobile ad hoc networks. In *Proceedings of the 6th annual international conference on Mobile computing and networking*, MobiCom '00, pages 255–265, New York, NY, USA, 2000. ACM.

[9] B. Nath and D. Niculescu. Routing on a curve. *SIGCOMM Comput. Commun. Rev.*, 33(1):155–160, 2003.

[10] D. Niculescu and B. Nath. Trajectory based forwarding and its applications. In *Mobicom 03*, New York, NY, USA, 2003. ACM.

[11] N. Priyantha, H. Balakrishnan, E. Demaine, and S. Teller. Mobile-assisted localization in wireless sensor networks. In *INFOCOM 2005*, volume 1, pages 172 – 183 vol. 1, march 2005.

[12] R. C. Shah, S. Roy, S. Jain, and W. Brunette. Data mules: modeling a three-tier architecture for sparse sensor networks. In *Proceedings of the First IEEE International Workshop on Sensor Network Protocols and Applications*, pages 30–41, 2003.

[13] T. Small and Z. Haas. The shared wireless infostation model - a new ad hoc networking paradigm (or where there is a whale, there is a way). In *ACM MobiHoc*, pages 233–244, 2003.

[14] A. Somasundara, A. Ramamoorthy, and M. Srivastava. Mobile element scheduling for efficient data collection in wireless sensor networks with dynamic deadlines. In *Proceedings of the 25th IEEE International Real-Time Systems Symposium*, pages 296–305, 2004.

[15] B. Tas and A. S. Tosun. Project web page for simulation results. `www.cs.utsa.edu/~tosun/trajectory/trajectory.html`. Accessed: 31 January 2015.

[16] B. Yu and B. Xiao. Detecting selective forwarding attacks in wireless sensor networks. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, pages 8 pp.–, 2006.

A Closer Look into Privacy and Security of Chromecast Multimedia Cloud Communications

Ali Tekeoglu, Ali Şaman Tosun
Department of Computer Science
University of Texas at San Antonio
San Antonio, TX, 78249
Email: {icy449@my.utsa.edu, ali.tosun@utsa.edu}

***Abstract*—Chromecast is a small, system-on-chip device, that plugs into the HDMI port of a larger screen and turns it into a smart screen. It is designed for multimedia streaming in a home-network environment. By setting up Chromecast, you can stream videos onto a larger screen and control it from a mobile device such as a smart-phone, tablet or a laptop. The idea is to cast the multimedia to a larger second screen and use the smaller one as a remote control. Since its based on Google Cast SDK which is open to developers, a growing number of multimedia content providers such as YouTube, Netflix, Hulu and HBO are offering applications to support Chromecast streaming for mobile operating systems. This device uses Discovery and Launch (DIAL) protocol, developed by YouTube and Netflix. We examined the network packets exchanged between the smaller remote control device and the Chromecast attached larger screen. While Chromecast encrypts most of the content, remote control device sends control packets to the remote servers in the clear-text, which makes it vulnerable to reply-attacks or session-hijacking attacks. Besides, data transmission pattern leak personal information outside of the home-network, raising privacy concerns. Network protocols used by Chromecast are investigated and known vulnerabilities are listed. A method to detect the existence of Chromecast behind a home-router is proposed.**

I. Introduction

With the recent improvements in networking and multimedia technology in the last couple of years, streaming high definition audio and video, whether its local or remote, became increasingly popular in a home-network environment. Many hardware and software companies have been stimulating the consumer market by developing innovative, convenient, cost effective multimedia devices that are easy to integrate to a home-network setting. Apple TV [8], Google TV [10], Roku Streaming Player [13], Netgear NeoTV [12], and Amazon Fire TV [7] are just a few of the most popular devices on the market. Apple TV lets their users stream 1080p HD content over the internet from the most popular video providers such as Netflix, Hulu Plus, YouTube, Vimeo and iTunes. It has AirPlay technology to stream local content from mobile devices such as a phone or tablet. Google TV is a platform which is designed to combine entertainment from TV, apps, and the internet. It comes integrated into new smart TVs like LG Smart TV as well as with set-top-boxes such as NetGear NeoTV, Sony Internet Player, Asus Cube and Vizio Co-Star. Roku is in the market with a Chromecast like streaming HDMI stick that has similar hardware and functionality. In addition, Roku has set-top-boxes Roku 1, Roku 2 and Roku 3 which are comparable to Apple TV. Netgear NeoMediacast HDMI Dongle [11] is another HDMI stick, which is about to be released to the market. It runs Android OS and is decorated with the latest specification WiFi hardware, Miracast functionality and integrated DRM support. Chromecast is a cost effective way of streaming multimedia, either from Internet or locally stored, onto a larger screen. Its an HDMI stick that connects to HDMI port and communicates over Wi-Fi. Among the severe competition in the market, Chromecast [9] has emerged as the most successful with respect to functionality/cost metric.

In this paper, we have evaluated the network protocols used by Chromecast; packet flow between client, Chromecast and external servers; and examined the network communications in a blackbox manner from security and privacy perspective.

The rest of the paper is organized as follows: Section II discusses the related work. In Section III, background information about the Chromecast Device and DIAL protocol is presented. Experimental test-bed setup and experimental results are given in Section IV. Conclusion of the paper and future work plans about security of Chromecast is presented in Section VI.

II. Related Work

In this section, the related work on security analysis of protocols used by Chromecast and the relevant papers will be briefly mentioned. To the best of our knowledge, there has been no evaluation of the Chromecast device in the literature.

Chromecast runs a stripped-down version of Android OS, with limited capability of application installation. Although there has been a flurry of research on Android security recently, most of the research focuses on full-fledged Android OS and application security. More than 1000 Android malware samples were investigated in [24] and most of the malware were repackaged versions of legitimate applications with malicious payloads. Different approaches were used to investigate security of Android applications. Taintdroid [3] monitors behavior of Android applications to detect misuse of private information and Kirin [5] is a security service for lightweight certification to mitigate malware at install time. Static analysis was used on more than 1,000 Android applications for security [4] and widespread misuse of private information and advertising networks is detected. Readers are referred to [6] for an overview of Android Security Framework. All of the above research focuses on a complete

Android OS and to our knowledge no work exists for stripped-down android OS used on Chromecast device.

STUN protocol is used by Chromecast while a local computer with a Chrome browser running TabCasting extension to mirror a browser tab onto Chromecast. Recently, researchers have uncovered multiple vulnerabilities in Belkin WeMo Home Automation devices, one of which was due to the way STUN protocol used in the framework [14]. According to the advisory, using one Wemo device an attacker may be able to use the STUN & TURN protocols to relay connections to any other Wemo device. There are possible attack scenarios listed in the Security Considerations section of RFC-5389 [21]. Attacks against the STUN protocol include, outside attacks; where an attacker modifies the messages in transit in order to fail the operation of STUN between client and server. Inside attacks could occur, such as a rogue client that may try to launch a DoS attack against a server by sending large number of STUN requests. In security sensitive applications TLS might be used to encrypt the STUN packets, however Chromecast does not make use of TLS with STUN. To the best of our knowledge, there is no published academic paper in the literature discussing the attacks scenarios mentioned in the RFC in detail.

Network Time Protocol (NTP) is used by Chromecast to synchronize its time after a reboot or when connecting to the internet after an extended period of time. In [15], authors discuss the security issues of NTP protocol. As a solution to address the security problems in NTP protocol, authors proposed a new security model named ***Autokey*** which is implemented in NTP version 4. However, in our experiments NTP client included with Chromecast's operating system always used the NTP version 3, which has many known vulnerabilities and security holes.

Our experiments made use of commodity hardware running on Linux OS, in order to capture the Chromecast traffic. The work of authors in [1] provides guidelines for configuring the packet capturing system for optimal performance. Even though, in our experiments we did not stress the network with more than normal amount of traffic, the methods in this paper are applied to improve the packet capture performance for the benefit of future work experiments such as DoS resilience of Chromecast.

III. Background

In this section we will discuss the underlying protocol Chromecast device and Chromecast enabled apps are talking in order to discover each other.

A. Discovery and Launch (DIAL) Protocol

Netflix and YouTube engineers developed Discovery and Launch (DIAL) Protocol [17] in cooperation. This open source protocol eases the application development and integration.

This protocol is used by Chromecast, for the purpose of discovering and launching applications on 1st screen devices, by 2nd screen devices. 1st screen devices have smaller screens, such as a tablet, mobile phone or a laptop. Whereas, 2nd screen devices are primarily built for better display with larger screen sizes. TVs, set-top boxes, Blu-ray players, LCD monitors could be classified as 2nd screen devices.

In a situation where one finds a video on his/her mobile phone and wants to play it on a local connected TV, he can launch the mobile app on the phone and then tap the Play on TV button on the mobile app. DIAL protocol is the underlying communication protocol that enables this simple interaction [17].

DIAL is only used for discovery and launch of an application, it does not provide further means of communication between 2nd and 1st screen apps. DIAL also, does not involve nor require pairing or authentication, since its a simple mechanism for discovering and launching apps on a single subnet, such as a home network [17].

Chromecast and devices on the same network with Chromecast communicates via DIAL protocol to discover each other. Both devices send multicast SSDP messages over UDP to IP address 239.255.255.250 on port 1900. *M-SEARCH * HTTP/1.1* packets are multicasted by clients for Chromecast discovery, while Chromecast device sends *NOTIFY * HTTP/1.1* packets to announce the IP address and port its listening to and the location of its device description XML file. According to *NOTIFY* * packets sent by Chromecast, device description file is found at http://local-chromecast-ip-address:8008/ssdp/device-desc.xml. Chromecast runs a UPNP Server with a kernel linux/3.8.13, over UPnP version 1.0 and it uses libupnp1.6.18 Portable SDK for UPnP devices. X-User-Agent field has *redsonic* in Chromecast SSDP packets.

One security concern mentioned in DIAL specification involves properly dealing with optional DIAL payloads. DIAL payloads are a container for information that can be passed to a first screen app via the DIAL start request as URL encoded UTF-8 strings. If payloads are accepted, 1st screen app-develeper must ensure to do a proper security analysis since these requests could come from an untrusted source [17].

IV. Experimental Results

In this section, we provide and discuss the results of experiments executed on the testbed shown in Figure 1.

Fig. 1. Experimental Network Setup

A. Home-Network Testbed Setup

In our experimental test-bed we had; a Chromecast device plugged into the HDMI port of an LCD monitor, an Android tablet as Chromecast remote device, and a laptop as a traffic monitoring device are connected to a D-Link DIR-615 v.E3 Wireless Router as shown in Figure 1.

The router used in the experiments was a D-Link DIR-615rev.E3 which is fairly cheap, low-end, targeted for home-user, off-the shelf device. Router hardware is flashed with open-source Open-Wrt [19] firmware. Open-Wrt [19] is a Linux based alternative OpenSource firmware suitable for a great variety of WLAN routers and embedded systems. We made use of *iptables* [16] firewall that is embedded into most linux based kernels. Open-Wrt kernel comes with iptables firewall.

Even though linux kernels include *iptables*, because of memory space limitations, open-wrt excluded the module required for port-mirroring functionality. We have customized and built the source after adding the support for iptables kernel module ipt_TEE. With the ipt_TEE support, it was possible to mimic the high-end router capability called "Port-Mirroring" with our low-end router.

By adding rules to *mangle* [23] table of iptables firewall, to send a copy of each packet destined or sourced from the Chromecast device IP, we were able to monitor the network traffic.

The following *iptables* commands are used to mirror all of the network communication;

```
$ iptables -t mangle -A POSTROUTING -s 192.168.1.0/24 -j TEE --gateway 192.168.1.154
$ iptables -t mangle -A POSTROUTING -d 192.168.1.0/24 -j TEE --gateway 192.168.1.154
```

The first firewall rule here, sends a copy of all locally generated packets to the gateway machine at IP address 192.168.1.154, which is running wireshark to capture the packets.

Second rule mirrors a copy of each packet that is destined to a local machine, coming from an outside machine.

In order to connect to Chromecast, Google's official Chromecast app is installed to the Android tablet from Google PlayStore. Additionally, couple of apps such as HBO Go, YouTube, Pandora, Netflix, HuluPlus, RedBullTV and Vevo are installed on the tablet device.

Both Chromecast and Tablet are connected through wireless to the D-Link Wi-Fi Router. The network traffic monitoring device is connected to the router via an ethernet cable to enhance the packet forwarding performance of router and not to effect the wireless spectrum capacity that is used for streaming with Chromecast and Tablet device.

B. Experiments, Results & Findings

In our experiments, we have installed Chromecast App Software Version 1.7.4 on our Nexus-7, Android 4.4.4 tablet. The Chrome browser installed on desktop, used Google-Cast plug-in version 14.805.0.6 for TabCasting experiments. Chromecast device itself is automatically checking for updates each time it boots up, the latest firmware version during our experiments was 17977. By examining the mirrored/captured network packets, we wanted to investigate the connections that are opened, kept-alive, protocols that are used, packet flow between Chromecast sender apps (resides on the small screen device) and the Chromecast receiver app. We have found several interesting points that might be either improved or extended for better privacy of Chromecast users. Following are the different experiments that have been done to examine the network packets exchanged under different scenarios.

- ***Experiment#1- Tablet device casting YouTube video to Chromecast:*** YouTube app in Anroid tablet has a cast to Chromecast button enabled, if the mobile device has previously discovered a Chromecast device in the same network. Once tapped, the tablet would stop showing the video in the small screen and start streaming the YouTube video to the Chromecast attached larger screen. Tablet device becomes a remote control. In this setting the control packets such as play, stop, pause etc. are sent through clear-text over HTTP from tablet to YouTube servers. This would pose a security threat and provide feasibility for man-in-the-middle attacks and replay-attacks. The video control packets are sent from the mobile remote device to YouTube servers and video is streamed directly to the Chromecast. When looked closer to the control packets; they use HTTP POST, HTTP GET methods to control the video played on Chromecast attached screen. Several information that could be extracted from these packets that are going unencrypted such as; the google account username (if logged in to YouTube), which video is being watched at what time of the day, what kind of Operating System and which version is installed on the remote device (Android 4.2.2, iOS etc), brand and model of remote device used (brand=Asus,model=Nexus) etc. Even if listened passively from outside home-network without any attacks, this information could be seen as a leak of privacy.
- ***Experiment#2- Vulnerability Scanning Chromecast:*** OpenVAS (Open Vulnerability Assessment System) [20] is an open source vulnerability scanner that is updated daily from several security advisories. As of writing this paper, it has about 35000 NVTs (Network Vulnerability Tests) in its database. There are multiple components in OpenVAS Framework; OpenVAS Manager, OpenVAS Scanner and Web Interface. In our testbed, OpenVAS-7 is installed in a virtual machine running on an Ubuntu host, connected in the same local network as Chromecast. We run the most rigorous test against Chromecast to scan it for known vulnerabilities. However, OpenVAS couldn't find any in Chromecast. Nmap scan listed 3 open ports before the setup in Chromecast; 53, 8008 and 8009. Port 53 is listening for DNS queries while Port 8008 is listening for http connections and port 8009 is listed as "ajp13" service. Apps are communicating with the device through Port 8008.
- ***Experiment#3- Cipher Suites used by Chromecast de-***

vice: Chromecast uses TLSv1.2 for communicating with TLS capable servers. For example; Chromecast communicates over TLSv1.2 while downloading an image to display during idle. After getting the IP address of the content server from Google's DNS, Chromecast and the content server does the 3-way TCP Handshake which is followed by Client Hello and Server Hello TLS messages. In our experiment which spanned around 65 hours, at idle, all the internet connections to the remote servers which are shown at the Figure 8 utilized TLS v1.2, except NTP servers and google's DNS server.

In the TLS v1.2, after handshake, client and server sends Client Hello and Server Hello Messages in order. In Client Hello messages, client offers a list of Cipher Suites that it supports. Each Cipher Suite defines the key exchange algorithm for authentication, as well as the subsequently used symmetric encryption for bulk encryption and integrity check algorithms for message authentication code calculation. Server responds with a Server Hello message in which it chooses one of them out of the client supported cipher suites list [2]. In our idle experiment that spanned 65 hours, Chromecast offered two different Cipher Suite list in its Client Hello messages. Lists either consisted of 18 or 62 different suites. About 99% of the Client Hello messages consisted of 18 cipher suites, where the tiny 1% offered 62 different cipher suites.

Out of the list of Cipher Suites supported by Chromecast, the most chosen by Google Server's was ***TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256***, which was also listed as the first cipher suite in the Chromecast's list, by 99.29 % of the time. ***TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256*** and ***TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384*** was also chosen by some servers. In our 65 hour idle experiment, most popular cipher suite picked up by servers 2123 times out of 2138 times.

- ***Experiment#4- Remote server orchestrated local Chrome TabCasting:*** Chrome TabCasting is an experimental future that is used through the GoogleCast Plugin to Chrome Browser. Chrome Browser extensions/plugins are installed from the Chrome Web Store. After installing this extension on a desktop Chrome Browser, browser can cast the view of a tab to a local ChromeCast device if one is found on the same local network. The interesting problem here that was found after capturing the network packets is that the desktop casting the Chrome Tab is connecting to a remote STUN [21], [22] server and sending the cast through it instead of just sending the packets directly to the Chromecast device locally.

  In this experiment we have monitored the CPU and memory usage of a TabCasting machine. We have used *sysstat* tools (pidstat) to capture the CPU and Memory usage of all the processes that Chrome Browser runs. Figure 3 shows the CPU usage of the chrome processes while tab-casting. Figure 2 shows the total Memory

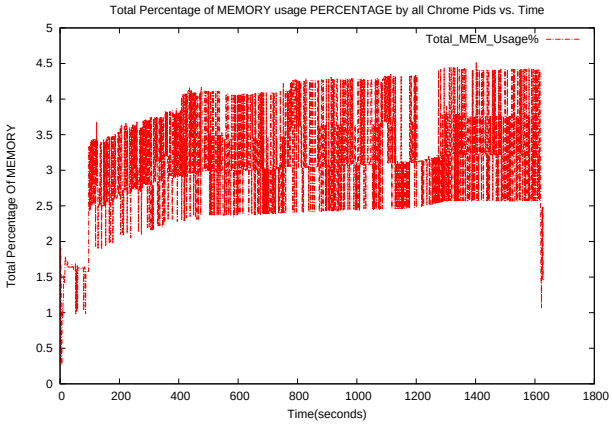


Fig. 2. Memory usage percentage of Desktop machine while TabCasting to Chromecast

usage of the Chrome processes while tab-casting. As figures show, TabCasting is overly resource consuming, especially for CPU cycles. CPU cycles are calculated with combining total usage of 11 processes belongs to Chrome browser; which are assigned to run on different CPU cores in parallel, thus total CPU usage spikes over 100% in Figure 3. Even though its quite useful for multimedia websites that do not a have a Chromecast app yet, this feature is still experimental and not even close to optimal. A router with simultaneous dual band Wi-Fi interfaces (2.4 and 5 GHz); one for Chromecast, other for the desktop would yield much better performance.

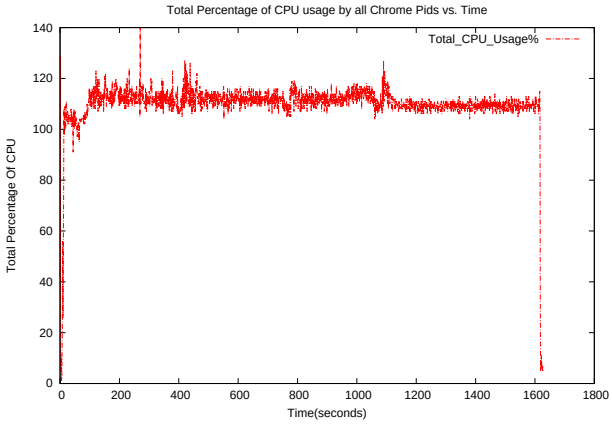


Fig. 3. CPU performance on the Desktop while tabcasting to Chromecast

- ***Experiment#5- Connections maintained when Chromecast is running in idle:*** Chromecast maintains a couple of connections to Google servers when it is connected to the home wireless network, but not streaming video. One connection is for retrieving the images that are displayed on screen as a wallpaper. For this purpose, Chromecast connects to a Google server every 60 seconds to download the next image.

  These images are retrieved from several different Google servers. Chromecast queries the Google's DNS server at 8.8.8.8, which in turn responds with couple of different IP

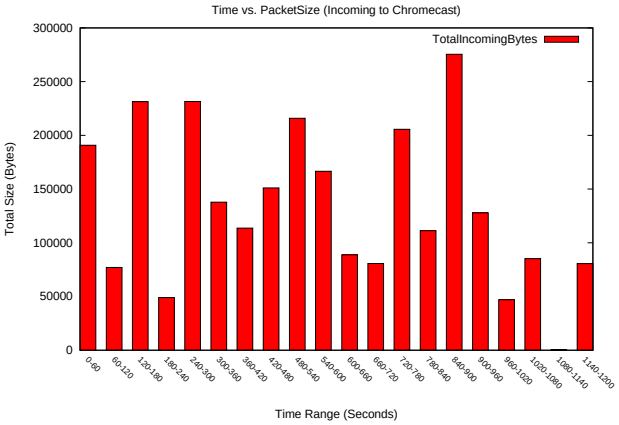


Fig. 4. Bar Graph of Incoming Packets to Chromecast at Idle

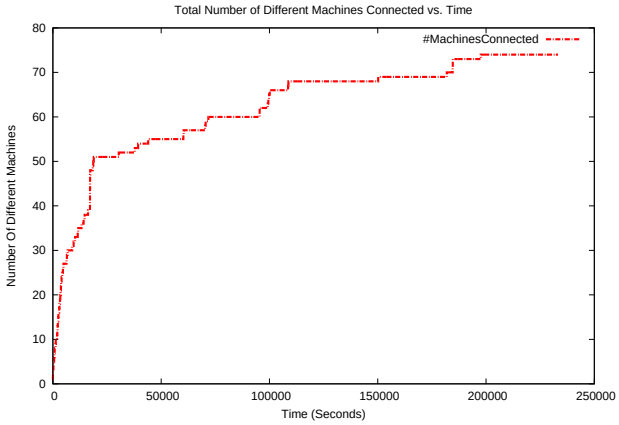


Fig. 5. Chromecast connecting to different machines at idle

addresses of Google content servers that Chromecast can connect and download a new image to display as wallpaper. Chromecast opens a connection to the first address in the DNS response. After a successful TCP-Handshake they set-up a TLS v1.2 connection, with Chromecast offering Cipher Suites for authentication-bulk encryption-decryption-Hashing and signature. In case Chromecast would keep downloading the image from the IP address, it keeps the connection alive with a TCP Keep-Alive packet, which is sent 45 seconds after wallpaper image is downloaded from the server. 15 seconds after the TCP Keep-Alive, another image is requested from the same server or DNS is queried again to get another IP to keep downloading wallpaper images.

The bar-graph in Figure 4 shows us the total amount of data incoming to Chromecast device per minute. This corresponds to the wallpaper images that are displayed on screen when the Chromecast is not used actively by another local remote control device. Although the packets are encrypted with TLS v1.2, someone listening the traffic from outside can differentiate which image is displayed on the screen from the total size of incoming packets every 60 seconds.

Data from a 3 days of network trace plotted in Figure 5 shows that Chromecast, when running in idle, is connecting to many different IP addresses. Even though its connecting to 11 different domains as shown in Figure 8, every once in a while IP address corresponding to the same domain name changes for load balancing & availability reasons. This could be mistaken as a malware leaking information from internal network over-time.

Figure 6, shows us the number of queries per domain name sent by Chromecast to DNS server during a streaming session of 2 hours from YouTube. When Figure 6 and Figure 8 are compared, YouTube streaming session of only 2 hours establishes connections to more domains as opposed to days of idle streaming.

To examine the total number of different IP address that Chromecast connects during the same 2 hours YouTube

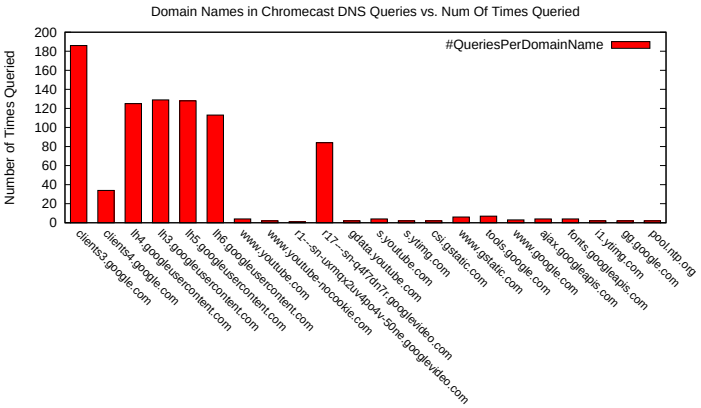


Fig. 6. Chromecast DNS Queries to Google DNS server while streaming from YouTube

streaming experiment; Figure 7 is plotted. Chromecast connects to several Google content servers on different IPs for image download on idle as seen in Figure 5 and for downloading different parts of stripped video as depicted in Figure 7 while streaming from YouTube.

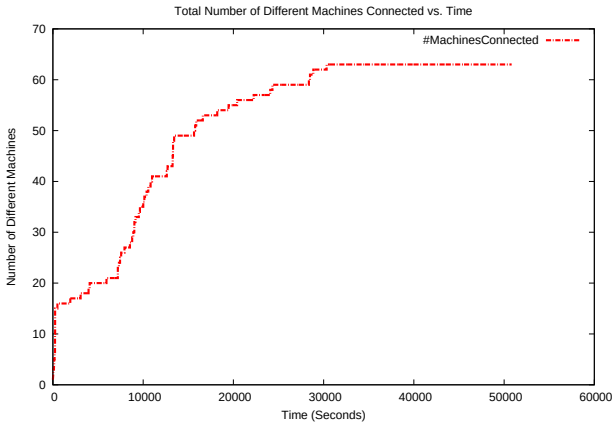


Fig. 7. Chromecast connecting to different machines while streaming from YouTube

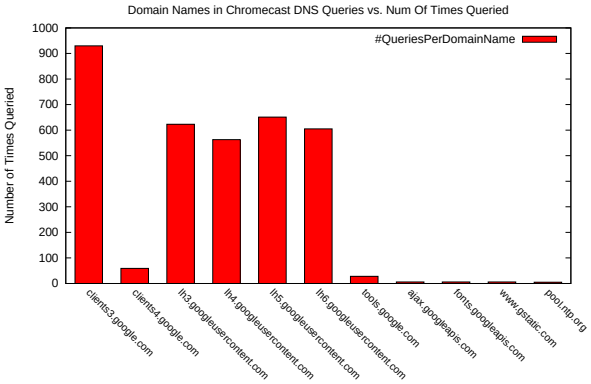


Fig. 8. Chromecast DNS Queries at idle to Google DNS server

- ***Experiment#6- DNS Queries at idle:*** Chromecast always uses the Google DNS Server, 8.8.8.8. As show in Figure 8 the requests at idle are for the following Address Mapping Records(A);
  For random wallpaper image download at idle, first Chromecast queries the 8.8.8.8 Google DNS server for an Address Mapping Record(A) with the the name lh3.googleusercontent.com, lh4.googleusercontent.com, lh5.googleusercontent.com or lh6.googleusercontent.com. Then, DNS server replies with couple of IP addresses which maps to the cannonical name (CNAME) googlehosted.l.googleusercontent.com. Also, Address Mapping Records(A) for clients3.google.com and clients4.google.com are queried from DNS server and it replies with usually 17 IP records for canonical name clients.l.google.com. These are also used for wallpaper image download.
  Another DNS query Chromecast sends to 8.8.8.8 asks for the IP address of tools.google.com, which the DNS server replies with 17 IP addresses mapped to the Canonical Name Record (CNAME) of tools.l.google.com. However, this query is not common, in our experiments it was queried once in every 3-4 hours.
  Chromecast queries the DNS server for pool.ntp.org address. DNS server replies with 4 random different IP addresses. In our experiments the interval for checking the ntp servers for Chromecast is not frequent. When Chromecast first checked the time, its clock has about 2 seconds of time drift from the server. There is not regular pattern for Chromecast to check for time.
  Another DNS query was for the ajax.googleapis.com which returned an IP for the CNAME googleapis.l.google.com. This query was made only once during our 1 day experiment. Another one time query was for www.gstatic.com.
- ***Experiment#7: NTP (Network Time Protocol):*** Chromecast uses NTP protocol after it's rebooted to get the current time through a NTP server. However, there are some known vulnerabilities in this protocol, specifically the version that is used by the servers that Chromecast is connecting for time synchronization. The most current stable version of NTP is listed as 4.2.6 at ntp.org official website [18] which has been released in 2011. When Chromecast queries the Network Time Server, it uses NTP Version 3 which was released in 1999 that has long been deprecated. Even though Chromecast uses an insecure version of NTP, this connections are infrequent, as seen in the experiment it has connected to an NTP server only 5 times over a 65 hours of idle period.

V. Discussion

Chromecast Detection: During our experiments, some patterns of network behaviour are found to be specific to Chromecast. There is not one simple formula to prove the existence of Chromecast behind a home-router, however the following could be combined to come up with a tool to detect it.

a) *MAC address of Chromecast:* which has the prefix of (6C:AD:F8), traces back to AzureWave Technologies, Inc.
b) *Chromecast Hot Spot* broadcasts if its powered up but not connected to the home Wi-Fi router. Which is also dangerous because anyone nearby searching for Wi-Fi routers can connect to it and change its settings.
c) *Image downloading from Google servers* with a 60 second interval is a characteristic behaviour of Chromecast at idle.
d) *TLS v1.2 Client Hello* message lists almost always 18 Cipher Suites that Chromecast supports.
e) *TabCasting future* uses the STUN protocol, however, instead of using registered UDP port number 3478 for this protocol, Chromecast communicates over port 19302 with Google STUN server which is for Google Talk Voice and Video connections.

These are not exclusively characteristic features of Chromecast but they could be combined together to implement a tool that can accurately detect Chromecast behind a NAT device. ISP's could make use of this tool to detect their users who are streaming videos onto a Chromecast behind their home-router.

VI. Conclusion and Future Work

We examined the network packets exchanged between the smaller remote control device and the Chromecast attached larger screen. While Chromecast encrypts most of the content, remote control device sends control packets to the remote servers in the clear-text, which makes it vulnerable to several attacks. Besides, it leaks personal information of the user to outside network, raising privacy concerns. We have listed the protocols used by Chromecast and their known vulnerabilities. We also proposed a simple method to detect the existance of a Chromecast device behind a home-network.

Future work includes but not limited to; implementation of Chromecast detection tool, a tool for injecting control packets like stop, pause, restart, for ongoing YouTube sessions and casting new videos to Chromecast attached screen, executing a DoS attack based on STUN protocol weaknesses, examining network communication of official Chromecast supported apps

on the market, and comparing the security of other HDMI streaming sticks from Amazon, Netgear and Roku.

REFERENCES

[1] Lothar Braun, Alexander Didebulidze, Nils Kammenhuber, and Georg Carle. Comparing and improving current packet capturing solutions based on commodity hardware. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, IMC '10, pages 206–217, New York, NY, USA, 2010. ACM.

[2] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), August 2008. Updated by RFCs 5746, 5878, 6176.

[3] William Enck, Peter Gilbert, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N. Sheth. Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, OSDI'10, pages 1–6, Berkeley, CA, USA, 2010. USENIX Association.

[4] William Enck, Damien Octeau, Patrick McDaniel, and Swarat Chaudhuri. A study of android application security. In *Proceedings of the 20th USENIX Conference on Security*, SEC'11, pages 21–21, Berkeley, CA, USA, 2011. USENIX Association.

[5] William Enck, Machigar Ongtang, and Patrick McDaniel. On lightweight mobile phone application certification. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, pages 235–245, New York, NY, USA, 2009. ACM.

[6] William Enck, Machigar Ongtang, and Patrick McDaniel. Understanding android security. *IEEE Security and Privacy*, 7(1):50–57, January 2009.

[7] Amazon Inc. Amazon Fire TV. "www.amazon.com/FireTV", retrieved, December 2014.

[8] Apple Inc. Apple TV. "http://www.apple.com/appletv/", retrieved, December 2014.

[9] Google Inc. Chromecast. "www.google.com/chromecast/", retrieved, December 2014.

[10] Google Inc. Google TV. "http://www.google.com/tv/", retrieved, December 2014.

[11] Netgear Inc. NeoMediacast HDMI Dongle. "http://www.netgear.com/service-providers/products/connected-media/set-top-boxes/NTV300D.aspx", retrieved, December 2014.

[12] Netgear Inc. Netgear Streaming Players. "http://www.netgear.com", retrieved, December 2014.

[13] Roku Inc. Roku Streaming Player. "http://www.roku.com/", retrieved, December 2014.

[14] IOActive. Belkin WeMo Home Automation Vulnerabilities. "http://www.ioactive.com/pdfs/IOActive_Belkin-advisory-lite.pdf", retrieved, August 2014.

[15] David L. Mills. A Brief History of NTP Time: Memoirs of an Internet Timekeeper. *SIGCOMM Comput. Commun. Rev.*, 33(2):9–21, April 2003.

[16] netfilter.org. The netfilter.org IPTables Project. "http://www.netfilter.org/projects/iptables/index.html", retrieved, January 2014.

[17] Netflix and YouTube. DIAL (DIscovery And Launch) Protocol. "http://www.dial-multiscreen.org/", retrieved, November 2013.

[18] ntp.org. Network Time Protocol. "http://support.ntp.org", retrieved, December 2014.

[19] Open-Wrt. Open-Wrt, Wireless Freedom. "https://openwrt.org/", retrieved, December 2014.

[20] OpenVAS. Open Vulnerability Assessment System. "http://www.openvas.org/", retrieved, August 2014.

[21] J. Rosenberg, R. Mahy, P. Matthews, and D. Wing. Session Traversal Utilities for NAT (STUN). RFC 5389 (Proposed Standard), October 2008.

[22] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy. STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs). RFC 3489 (Proposed Standard), March 2003. Obsoleted by RFC 5389.

[23] thatexplainsalot.com. Use Wireshark And DD-WRT Router Firmware To Imitate Port Monitoring On A Router Switch Port. "http://thatexplainsalot.com/blog/2010/11/use-wireshark-and-dd-wrt-router-firmware-to-imitate-port-monitoring-on-a-router-switch-port/", retrieved, January 2014.

[24] Yajin Zhou and Xuxian Jiang. Dissecting android malware: Characterization and evolution. In *IEEE Symposium on Security and Privacy*, pages 95–109, 2012.

Blackbox Security Evaluation of Chromecast Network Communications

Ali Tekeoglu, Ali Şaman Tosun
Department of Computer Science
University of Texas at San Antonio
San Antonio, TX, 78249
Email: {icy449@my.utsa.edu, ali.tosun@utsa.edu}

***Abstract*—Chromecast is a small, system-on-chip device, that plugs into the HDMI port of a larger screen and turns it into a smart screen. It is designed for multimedia streaming in a home-network environment. By setting up Chromecast, you can stream videos onto a larger screen and control it from a mobile device such as a smart-phone, tablet or a laptop. We examined the network packets exchanged between the smaller remote control device and the Chromecast attached larger screen. While Chromecast encrypts most of the content, remote control device sends control packets to the remote servers in the clear-text, which makes it vulnerable to reply-attacks or session-hijacking attacks. Besides, data transmission pattern leak personal information outside of the home-network, raising privacy concerns. Network protocols used by Chromecast are investigated and known vulnerabilities are listed. A method to detect the existence of Chromecast behind a home-router is proposed.**

I. Introduction

With the recent improvements in networking and multimedia technology in the last couple of years, streaming high definition audio and video, whether its local or remote, became increasingly popular in a home-network environment. Chromecast [1] is a cost effective way of streaming multimedia, either from Internet or locally stored, onto a larger screen. Its an HDMI stick that connects to HDMI port and communicates over Wi-Fi.

In this paper, we have evaluated the network protocols used by Chromecast; packet flow between client, Chromecast and external servers; and examined the network communications in a blackbox manner from security and privacy perspective.

The rest of the paper is organized as follows: Experimental testbed setup and experimental results are given in Section II. We discuss a method to detect Chromecast in Section III. Conclusion of the paper is presented in Section IV.

II. Experimental Results

In this section, we provide and discuss the results of experiments executed on the test-bed shown in Figure 1.

A. Home-Network Testbed Setup

In our experimental testbed we had; a Chromecast device plugged into the HDMI port of an LCD monitor, an Android tablet as Chromecast remote control device, and a laptop as a traffic monitoring device that are all connected to a D-Link DIR-615 v.E3 Wireless Router as shown in Figure 1.

Fig. 1. Experimental Network Setup

Router hardware is flashed with open-source Open-Wrt [4] firmware. Open-Wrt [4] is a Linux based alternative OpenSource firmware suitable for a great variety of WLAN routers and embedded systems. We made use of *iptables* [2] firewall that is embedded into most linux based kernels. Open-Wrt kernel comes with *iptables* firewall, however, because of memory space limitations, Open-Wrt excluded the module required for port-mirroring functionality. We have customized and built the source after adding the support for iptables kernel module ipt_TEE. With the ipt_TEE support, it was possible to mimic the high-end router capability called "Port-Mirroring" with our low-end router.

By adding rules to *mangle* table of iptables firewall, to send a copy of each packet destined or sourced from the Chromecast device IP, we were able to monitor the network traffic.

The following *iptables* commands are used to mirror all of the network communication;

```
$ iptables -t mangle -A POSTROUTING -s 192.168.1.0/24 -j TEE --gateway 192.168.1.154
$ iptables -t mangle -A POSTROUTING -d 192.168.1.0/24 -j TEE --gateway 192.168.1.154
```

The first firewall rule here, sends a copy of all locally generated packets to the gateway machine at IP address 192.168.1.154, which is running wireshark to capture the packets while second rule mirrors a copy of each packet that is destined to a local machine, coming from an outside machine.

B. Experiments, Results & Findings

By examining the mirrored/captured network packets, we wanted to investigate the connections that are opened, kept-

alive, protocols that are used, packet flow between Chromecast sender apps (resides on the small screen device) and the Chromecast receiver app. We have found several interesting points that might be either improved or extended for better privacy of Chromecast users.

- ***Experiment#1- Casting YouTube video to Chromecast:*** The control packets such as play, stop, pause etc. are sent through clear-text over HTTP from tablet to YouTube servers. This would pose a security threat and provide feasibility for man-in-the-middle attacks and replay-attacks. Unencrypted packets leaked information such as; google account username (if logged-in to YouTube), name and time of the video being watched, Operating System and it's version installed on the remote device (Android 4.2.2, iOS etc), brand and model of remote device (brand=Asus,model=Nexus) etc.
- ***Experiment#2- Vulnerability Scanning Chromecast:*** OpenVAS (Open Vulnerability Assessment System) [5] is an open source vulnerability scanner that is updated daily from several security advisories. We run the most rigorous tests against Chromecast to scan it for known vulnerabilities, however, OpenVAS couldn't find any. On the other hand, nmap scan listed 2 open ports in Chromecast; 8008 and 8009. Port 8008 is listening for http connections and port 8009 is listed as "ajp13" service.
- ***Experiment#3- Cipher Suites used by Chromecast:*** Chromecast offered two different Cipher Suite lists in its Client Hello messages. Lists consisted of either 18 or 62 different suites. About 99% of the Client Hello messages consisted of 18 cipher suites, where the tiny 1% offered 62 different cipher suites. This list of offered cipher suites could be used to detect a Chromecast behind a router.
- ***Experiment#4- Remote server orchestrated local Chrome TabCasting:*** The desktop casting the Chrome browser tab is connecting to a remote STUN [6] server and sending the cast through it instead of sending the packets locally, however, STUN protocol has several known vulnerabilities listed.
- ***Experiment#5- Connections maintained at idle state:*** Chromecast maintains a couple of connections to Google servers when it is connected to the home wireless network, but not streaming video. One connection is for retrieving the images that are displayed on screen as a wallpaper. For this purpose, Chromecast connects to a Google server every 60 seconds to download the next image. Although the packets are encrypted with TLS v1.2, someone listening the traffic from outside can differentiate which image is displayed on the screen from the total size of incoming packets every 60 seconds.
- ***Experiment#6- DNS Queries at idle state:*** Chromecast always uses the Google DNS Server, 8.8.8.8. For random wallpaper image download at idle, an Address Mapping Record(A) for *lh3.googleusercontent.com, lh4.googleusercontent.com, lh5.googleusercontent.com or lh6.googleusercontent.com, clients3.google.com and clients4.google.com* are requested. Other DNS queries asked for the IP addresses of *tools.google.com and pool.ntp.org*.In our experiments the interval for checking the ntp servers for Chromecast is not frequent. Another DNS query was for the *ajax.googleapis.com* This query was made only once during our 1 day experiment. Another one time query was for *www.gstatic.com*.
- ***Experiment#7: NTP (Network Time Protocol):*** Chromecast uses NTP [3] protocol for getting the current time through an NTP server each time it's rebooted. However, there are some known vulnerabilities in this protocol, specifically the version that is used by the servers that Chromecast is connecting for time synchronization.

III. Discussion

During our experiments, some patterns of network behaviour are found to be specific to Chromecast. The following could be combined to come up with a tool to detect it.

a) *MAC address of Chromecast* has the prefix of (6C:AD:F8), traces back to AzureWave Technologies, Inc.
b) *Chromecast Hot Spot* broadcasts if its powered up but not connected to the home Wi-Fi router. Which is dangerous because anyone nearby searching for Wi-Fi routers can connect to it and change its settings.
c) *Image downloading from Google servers* with a 60 second interval is a characteristic behaviour of Chromecast at idle.
d) *TLS v1.2 Client Hello* message lists almost always 18 Cipher Suites that Chromecast supports.
e) *TabCasting future* uses the STUN protocol, however, instead of using registered UDP port number 3478 for this protocol, Chromecast communicates over port 19302 with Google STUN server which is for Google Talk Voice and Video connections.

IV. Conclusion

We examined the network packets exchanged between the smaller remote control device and the Chromecast attached larger screen. While Chromecast encrypts most of the content, remote control device sends control packets to the remote servers in the clear-text, which makes it vulnerable to several attacks. Besides, it leaks personal information of the user to outside network, raising privacy concerns. We have listed the protocols used by Chromecast and their known vulnerabilities. We also proposed a simple method to detect the existance of a Chromecast device behind a home-network.

References

[1] Google Inc. Chromecast. "www.google.com/chromecast", retrieved, December 2013.
[2] netfilter.org. The netfilter.org IPTables Project. "http://www.netfilter.org/projects/iptables/index.html", retrieved, January 2014.
[3] ntp.org. Network Time Protocol. "http://support.ntp.org", retrieved, March 2014.
[4] Open-Wrt. Open-Wrt, Wireless Freedom. "https://openwrt.org/", retrieved, March 2014.
[5] OpenVAS. Open Vulnerability Assessment System. "http://www.openvas.org/", retrieved, August 2014.
[6] J. Rosenberg, R. Mahy, P. Matthews, and D. Wing. Session Traversal Utilities for NAT (STUN). RFC 5389 (Proposed Standard), October 2008.